.

Vom Fachbereich für Mathematik und Informatik
der Technischen Universität Braunschweig
genehmigte Dissertation
zur Erlangung des Grades eines Doktor-Ingenieurs (Dr.Ing) vorgelegt von

Armin Hopp

VR-LAB: A Distributed Multi-User Environment for Educational Purposes and Presentations

Datum der Promotion: 1.11.2000
(Tag der mündlichen Prüfung) : 1.11.2000

1. Referent: Prof. Dr.techn. Dieter W. Fellner
2. Referent: Prof. Dr.-Ing. Friedrich Wahl
eingereicht am: 30.6.2000

Acknowledgment

I want to thank all members of the CG group at the University of Bonn and the University of Braunschweig for their support. Especially Ralf Leonhard for his work on the project. Gordon Müller and Stephan Schäfer for their continuous help and the comments. Many thanks to Lydia Becker for proof reading. Additionally I want to thank Dirk Schulz from the University of Bonn for his support with the rhino robot. A special thank to Prof. Dr. Fellner for his encouragement and support through the years in Bonn and Braunschweig. A special thank to Elke.

Contents

1 Introduction **1**
- 1.1 Sample Scenario . . . 2
- 1.2 Overview . . . 3

2 Building Blocks **5**
- 2.1 Graphics . . . 5
  - 2.1.1 VRML . . . 5
  - 2.1.2 VRML97 EAI . . . 6
- 2.2 Data Distribution . . . 7
  - 2.2.1 HTTP/FTP . . . 7
  - 2.2.2 UDP Datagrams . . . 7
  - 2.2.3 RTP . . . 8
  - 2.2.4 Multicast . . . 8
  - 2.2.5 DIS . . . 9
  - 2.2.6 VRTP . . . 11
  - 2.2.7 LRMP, Lightweight Reliable Multicast Protocol . . . 12
  - 2.2.8 DWTP . . . 14
  - 2.2.9 "Unreal" NetworkingArchitecture . . . 15
  - 2.2.10 Discussion . . . 16
- 2.3 Data Transport indistributed Environments . . . 17
- 2.4 Corba . . . 18
- 2.5 Discussion . . . 18

3 Design and Implementation of MRT-VR **20**
- 3.1 Implementing aDistributed VRML System . . . 21
  - 3.1.1 Intention . . . 21
  - 3.1.2 Scenario . . . 22
  - 3.1.3 Access Rights . . . 22
  - 3.1.4 Interaction . . . 22
- 3.2 Distributed SceneDatabase . . . 23
  - 3.2.1 Participant and ViewpointManagement . . . 24
  - 3.2.2 Data Replication . . . 25
- 3.3 Data Transport . . . 30

3.3.1 TCP/UDP and TCP/IP . . . 30
3.3.2 TCX . . . 30
3.3.3 M-Bone . . . 31
3.3.4 Data Transport by Corba . . . 31
3.3.5 Data Transport by Multicast . . . 32
3.3.6 Selecting the Actual Transport Mechanism . . . 35
3.4 The MRT-VR System . . . 35
3.4.1 Collision Detection . . . 35
3.4.2 Fly Interactor . . . 38
3.4.3 Pointer Interactor . . . 38
3.5 Usability Enhancements . . . 38
3.5.1 Integration into SDR . . . 38
3.5.2 Camera Interpolation . . . 39
3.5.3 Coordinate SystemDetermination . . . 39
3.5.4 Frame Rate Detectionon Startup . . . 39
3.5.5 Avatar Selection . . . 40

4 Enhancements of the MRT Library 41
4.1 Enhancements of MRT . . . 41
4.1.1 Parsing Mechanism . . . 41
4.1.2 Control Objects . . . 41
4.1.3 DistribID . . . 42
4.1.4 Reference Objects . . . 42
4.1.5 Parser Extensions forCoupling of ExternalSimulations . . . 45
4.2 Architecture of MRT . . . 46
4.2.1 Objects . . . 46
4.2.2 Rays . . . 47
4.2.3 Shader . . . 47
4.2.4 Light Sources . . . 48
4.2.5 Camera Model . . . 48
4.2.6 Rendering . . . 48
4.2.7 Approximative Rendering . . . 48

5 Distributed Presentation Environments 49
5.1 Lecture Rooms . . . 49
5.1.1 IGD Darmstadt . . . 49
5.1.2 HNI, University of Paderborn . . . 50
5.1.3 University of Dortmund,Multimedia Lecture Room . . . 51
5.1.4 Discussion . . . 51
5.2 Design of a DistributedPresentation Environment . . . 51
5.2.1 Supported Media . . . 52
5.2.2 Video Media Quality . . . 53
5.2.3 Audio Media Quality . . . 53

5.2.4 Components Used . . . 54
5.3 User Interface Description . . . 54
5.3.1 Media Control . . . 54
5.4 Internet Remote Control of MMHS . . . 55
5.5 Usability Test . . . 56
5.5.1 Questionnaire . . . 58
5.5.2 Results . . . 58
5.5.3 Interpretation . . . 59
5.5.4 Conclusions . . . 60
5.6 VR-LAB Hardware Control . . . 61
5.6.1 Hardware Development . . . 61
5.6.2 Third Party Hardware . . . 64

6 3D-Projection 67
6.1 Stereoscopic Projection . . . 67
6.1.1 Active Projection Systems . . . 67
6.1.2 Passive Projection Systems . . . 67
6.1.3 Discussion . . . 68
6.2 An alternative Setup for3D Projection . . . 68
6.2.1 The Synchonisation-Separation-Circuit . . . 68
6.2.2 Alignment of PolarizationFilters . . . 70
6.3 Future Enhancements . . . 71

7 Sample Experiments 73
7.1 Sample Experiment:Robotic Simulation with the RTL . . . 73
7.1.1 The Robotic Tele Labor (RTL) System . . . 73
7.1.2 Auto Viewpoint Indicated by Simulation . . . 78
7.2 Sample Experiment:Particle Simulation . . . 80

8 Conclusion 82

9 Appendix 87
9.1 MMHS Control Hardware . . . 87
9.2 MMHS Control Software . . . 88
9.3 Hardware Control Codes for Third Party Hardware . . . 111
9.4 Hardware Control Codes for Custom Hardware . . . 112

References 112

VR-LAB: A distributed Multi-User Environment for Educational Purposes and Presentations

Dipl. Inform. Armin Hopp

January 12, 2001

Abstract

In the last three years our research was focused on a new distributed multi-user environment. Finally, all components were integrated in a system called the VR-Lab, which will be described on the following pages.

The VR-Lab provides Hard- and Software for a distributed presentation system. Elements which are often used in environments called Computer Supported Cooperative Work (CSCW).

In contrast to other projects the VR-Lab integrates a distributed system in a common environment of a lecture room and does not generate a virtual conference room in a computer system. Thus, allowing inexperienced persons to use the VR-LAB and benefit from the multimedia tools in their common environment.

To build the VR-LAB we developed a lot of hard- and software and integrated it into a lecture room to perform distributed presentations, conferences or teaching. Additionally other software components were developed to be connected to the VR-LAB, control its components, or distribute content between VR-LAB installations.

Beside standard software for video and audio transmission, we developed and integrated a distributed 3D-VRML-Browser to present three dimensional content to a distributed audience. One of the interesting features of this browser is the object oriented distributed scene graph. By coupling a high-speed rendering system with a database we could distribute objects to other participants. So the semantic properties of any geometrical or control object can be kept and used by the remote participant. Because of the high compression achieved by the transport of objects instead of triangles a lot of bandwidth could be saved. Also each participant could select a display quality appropriate to its hardware.

To enhance the features of the 3D-browser, the communication interface was made accessible to foreign applications, providing a high-end distributed 3D-visualization which can be easily coupled to other software, like simulators (e.g. rain or snow).

Additional work was spent on user navigation in 3D-space and immersive stereoscopic video displays to enhance the three dimensional effect. We developed a simple and low priced stereoscopic projection system, which is described in detail.

All components were evaluated by test persons and produced positive results.

Chapter 1

Introduction

Today's world of computers is more networked than ever before [Obl95][Sun93]. Important information may reach its destination within seconds, no matter where the recipient is situated. Almost every part of industry and business uses computers and networks to increase the efficiency of their staff. Besides the change of a single workplace, teamwork in the companies has changed a lot [Dam96, Pul96, Bar96].

In many cases networked computers are used for sharing and printing of documents. Other applications might support the workflow of documents. These environments are often called "Computer Supported Cooperative Work" (CSCW).

The Expression CSCW is often used in combination with the Internet because e-mail and WWW are used to distribute documents and information [RTS98]. Other authors may think of video-conferencing tools when talking about CSCW [Stu98].

During our research we found that the third dimension is more or less unknown in CSCW software, though Brutzman et al. [BZWM97] define network interactive 3D graphics as an essential human interface. Some applications use a 3D environment to simulate a conference room, but not to visualize any content [Stu98].

CSCW Systems share documents which are mostly word-processor, spreadsheets or (2D) presentations. Additionally we found that for specific tasks, such as video and audio conferencing, high quality tools are available, but the installation and realisation of a conference is often very complex.

Analysis has shown that there were a lot of 3D-browsers developed in the recent years. One reason for the rare use of three dimensional graphics may be that existing 3D browser are too difficult to operate [Leo96].

In contrast to other papers this work will not describe a single software component providing a CSCW environment like [Stu98] or [Bro97], but a complex system to control and provide distributed Video, Audio, 3D-graphics and stereoscopic images.

Besides this another important goal of our work was to provide an environment which simplifies the use of a distributed system and has an ergonomic, user friendly interface. If existing software components are to be used, their use should be simplified.

The VR-LAB consists of many different components, which is shown in the schematic overview of Figure 1.1. Some of the parts are hardware and some of them software. Some parts were integrated into the VR-LAB while others have to be newly developed: the presentation environment, its controlling software, the controlling components, their software and the distributed VRML-browser MRT-VR [1]. All

[1] MRT-VR: MRT (Modualar Rendering Toolkit), VR (Virtual Reality)

systems are capable of being controlled by a remote operator, allowing different modes of operation to suit the needs of different scenarios.

Main parts of the VR-LAB are:

- *video system:* video cameras, VCR, 35mm slides projectors, OHP, PC or workstation screen output and input, video mixing and video effects (MMHS [2])
- *audio media:* wireless microphones, amplifiers, speakers, sound mixer (MMHS)
- *network capabilities:* 100MBit-network, ATM
- *collaboration software (off the shelf):* SDR, VIC, RAT, Whiteboard, NetMeeting [MJ95, Cor96b] (MMHS)
- *Collaboration software (custom):* distributed VRML-browser MRT-VR, controlling software for the environment,
- *Control software (custom):* local MMHS software (Delphi), remote MMHS software (Java),
- *Control Software (MCU[3] custom):* controlling software for the environmental-, audio- and camera- controllers. (MMHS)

1.1 Sample Scenario

To give the reader an idea of the capabilities of the VR-LAB we will use a scenario [4], developed through our work at the University of Bonn and is illustrated in Figure 1.2.

We define a presentation or "session" as a situation where a presenter is talking about a specific content to a local and/or remote audience. The audience may be one or even more classes of users connected to the presenter and called "Participants". While talking about the content and using a distributed video and audio link, the presenter may want to show something which we will call an "experiment". In this scenario the example experiment deals with a "real" robot which is situated at another remote place controlled via a remote software. To control the robot's movements additional video links with fixed cameras might be used to provide the audience with the capture images of the experiment.

If we analyze this scenario in detail, some problems appear, which solutions are presented in this work.

To bring this scenario into life, a lot of the components of the VR-LAB can be used. At First, an environment is needed which provides a video projection and audio system for a distributed audience. Second, we need a system to control the capturing and distribution of images and sounds. This system might be operated local or remote by the presenter.

The experiment itself will raise other problems: a video link may not be the adequate tool to control a remote robot, because it may be too slow to be interactive. Another problem: the robot may move out of the camera's field of view.

To overcome those problems we can use a 3D-model and a simulation: if we replace the real world of the experiment by an approximation through the use of a VRML model and think of the robot as a part of it, make the whole system distributed and interactive, and even add a dead reckoning system to extrapolate transmission gaps, we will have a good tool to interactively watch the whole experiment from differ-

[2] Multi-Media-Hörsaal (MMHS), german for multi-media-lecture room

[3] Micro Controller Unit (MCU)

[4] Multi-Media-NRW Research. The MM-NRW research group consisted of 18 groups at German universities, financed by the country NRW, from May 1996 until the end of 1998

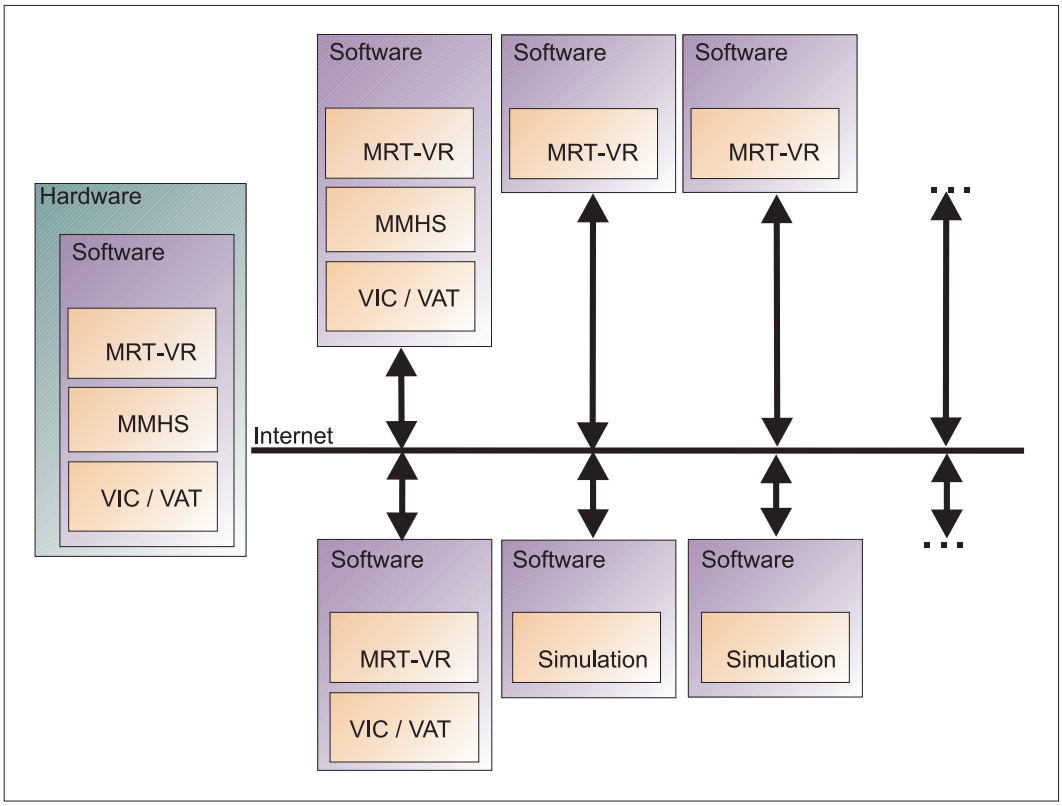

Figure 1.1: The Design of the VR-LAB

ent viewpoints at different locations in real-time.

Every participant can look at the model from a different viewpoint, while for a discussion the presenter will assure that all participants share the same view. To realize this we developed the distributed VRML-browser MRT-VR [HF99][HF98][HF96][HF97].

Additionally we developed a three dimensional projection system to generate a semi-immersive environment for the audience. An immersive system might be used to increase the three dimensional impression of VRML models when moving through a 3-dimensional world, like a virtual city or an architectural model.

This scenario will reappear at other places in this work to illustrate some of the problems and their solutions in more detail.

1.2 Overview

The VR-LAB and its components can be used to record, transmit or display sessions even if they are live on the network, from tape, or distributed. Operation can be controlled by the presenter or by a service operator in the lab or from remote. Integration of these systems allow audio, video, 2D and 3D interaction via the Internet.

In Chapter 2 we will give an overview of important software components of multi-user environments and their underlying transport protocols and discuss the problems arising when building distributed environments. Afterwards we will motivate the development of our system for distributed data transport and the appropriate implementation for a distributed VRML-

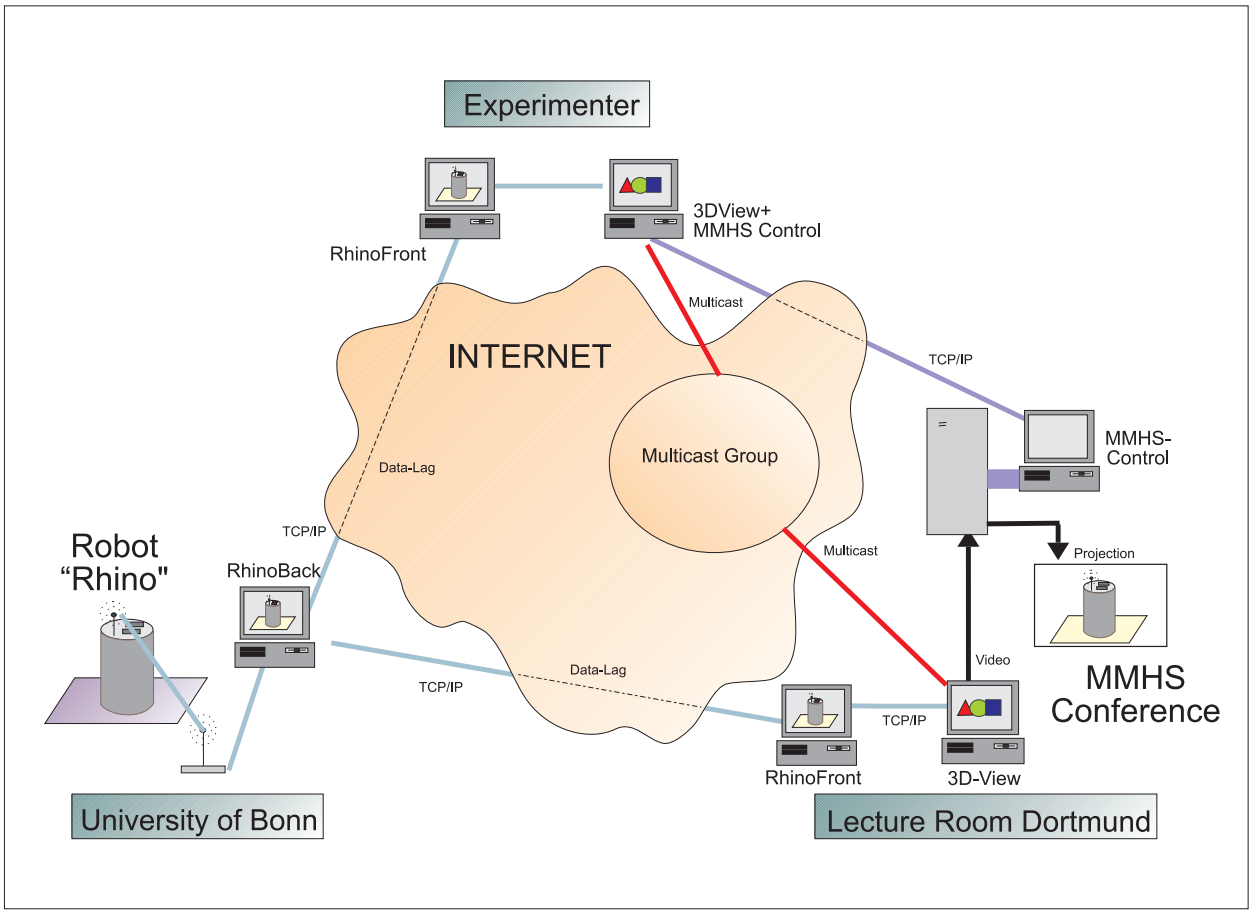

Figure 1.2: VR-LAB, the Big Picture

browser called MRT-VR (MRT- Virtual Reality) in Chapter 3.

Additional work was spent in extending the user interface of the system to enhance usability of 3D environments.

Chapter 4 gives a short overview about the MRT-Library (MRT: Modular Rendering Toolkit) which was used to realize MRT-VR and the implemented enhancements to the library. Chapter 5 describes the development of the presentation environment for the VR-LAB, called MMHS, which is motivated by the description of existing presentation environments and a discussion on their advantages and disadvantages.

Chapter 6 introduces the stereoscopic projection system developed for the MMHS to enhance the immersive effect for the audience. Chapter 7 describes some experiments which were performed using the VR-LAB and the MRT-VR software. Finally Chapter 8 gives the conclusion of our work.

Chapter 2

Building Blocks

This chapter describes the main building blocks which are used to build a distributed multi-user-system like the VR-Lab, and especially the distributed VRML-Browser MRT-VR which is used to show 3D content.

Besides the graphical description of content, which is mostly done by the use of VRML, the content has to be distributed to other participants, which is achieved by the use of different network transport protocols. A distributed database should be used to keep track of scene changes and the elements transported.

2.1 Graphics

One major part of our browser MRT-VR is three dimensional graphics to be presented to a distributed audience. One important standard for 3D worlds is VRML, which has become a standard for 3D only a few years ago, and unfortunately its further development is not clear. VRML 1.0 as a standard for 3D worlds only describes the content of the world, while VRML 2.0 brought 3D to life through the implementation of behaviors, realized by scripting in the VRML file. At the same time the VRML EAI (external authoring interface) was developed to modify the scene elements by programs external to the VRML script.

2.1.1 VRML

VRML has extended the Web to three dimensions (3D). Key contributions of the VRML 1.0 standard [BPP95a, BPP*95b] were a core set of object-oriented graphics constructs was used to build virtual worlds suitable for cross-platform 3D scene generation. VRML-based 3D Web browsers are usually embedded inside 2D browsers or are launched as helper applications when connecting to a 3D site. The typical network interaction model for VRML Web browser scenes remains client-server-HTTP via the browser, similar to non-3D Web browser and helper applications. Many interactive worlds have already been produced using the VRML specification. However, in order for VRML to scale to many simultaneous users, in addition to client-server query-responses, peer-to-peer interactions are necessary. Dynamic scene changes will need to be stimulated by a variable combination of scripted actions, message passing, user commands or entity behavior protocols (such as DIS). Thus the forthcoming VRML behaviors standardisation will need to simultaneously provide simplicity, security, scalability, generality and open extensions. For all these reasons we expect that networked VRML scenes will demand significant real-time streaming in addition to HTTP-based client-server interactions.

2.1.2 VRML97 EAI

The External Authoring Interface [CMB96] provides access to the nodes and fields of a VRML scene graph loaded by a 3D-browser (e.g. Netscape Navigator, MS-Internet Explorer) via Java. Using the EAI, field values can be changed via Java applications. The EAI provides access to ISO (IEC 14772-1). The Virtual Reality Modeling Language (VRML) defines a file format that integrates 3D graphics and multimedia. Conceptually, each VRML file is a 3D time-based space that contains graphics and aural objects [1] that can be dynamically modified through a variety of mechanisms. The EAI defines the interface that applications external to the VRML-browser may use to access and manipulate the objects defined in the VRML specification.

The EAI was designed to allow an external environment to access nodes in a VRML scene using the existing VRML event model. In this model an eventOut of a given node can be routed to an eventIn of another node. When the eventOut generates an event, the eventIn is notified and its node processes that event. Additionally, if a script in a script node has a reference to a given node it can send events directly to any eventIn of that node and it can read the last value sent from any of its eventOuts. The scope of the EAI was to cover all forms of access to a VRML-browser from external applications. The EAI does not provide a byte level description of the interface.

The EAI allows four types of access to the VRML scene:

- accessing the functionality of the browser script interface
- sending events to eventIns of nodes inside the scene
- reading the last value sent from eventOuts of nodes inside the scene
- getting notified when events change values of node fields inside the scene

As with scripts within the VRML scene, the EAI allows access to the full functionality of the browser script interface. The browser state can be queried, routes can be added and deleted, and new nodes can be created. The EAI extends the basic browser interface with a number of extra capabilities including retrieving node references and registering interest in browser events.

Though allowing access to most of the scene graph of a loaded VRML scene, the EAI has some disadvantages. Since EAI is only implemented for Java, communication to other applications is only possible by using Java. The security mechanisms of Java do not permit access to Multicast network services. When using a centralized server, all Java applications have to be started from the machine hosting the server application. Other disadvantages concern the access of browser state data, when retrieving object states. So the EAI does not allow to save the current state of a VRML scene. This is very important when users join a session already started, and it is not possible to send them the current state of the world. Some of these problems have been overcome by patching Java source code or adding additional code to save and restore current browser state information [Mau98]. Another disadvantage is the use of strings to create VRML nodes through the CreateVRMLFromString or createVRMLFromUrl command. To transmit scene changes the node data has to be transported in an ineffective way using an ASCII description. It is obvious that we lose information when transforming real numbers to string values.

[1] aural objects are objects which surround the user, objects that are near to the current user position.

2.2 Data Distribution

Connectivity between distributed applications is essential for transmitting information over the network. Because we have a mixture of different types of information to distribute over the Internet, different types of data transmission may be adequate. Due to the heterogeneity of data types, required bandwidth and reliability, a use of heterogeneous transport protocols is suggested rather than using one protocol for all types of data. Since there is no established infrastructure for efficient distribution available, developers of multi-user-virtual-environments (MVE's) are still dependent on low-level protocols.

We can distinguish three different types of data distribution via the network:

- *Unicast:* the sender sends a message exactly to one receiver which is known to the sender (1:1 point to point)
- *Broadcast:* the sender sends a message that may be received by all instances currently present in the system. Each receiver has to decide individually if a message is relevant. Broadcast packets may be filtered by network routers to avoid network pollution.
- *Multicast:* the sender sends messages to a well defined subset of instances present in the system.

Different protocols and their suitability for different applications were compared in [Saa98]:

Brutzman et al. [BZWM97] do not distinguish between strictly reliable and unreliable data transport and propose a more floating definition in their approach called VRTP:

Instead of having a floating definition we think that data transport should be distinguished by importance only. There are important and less important messages. Which type of data transport is sufficient, reliable or unreliable, should depend on the application and should be selected by the programmer. Future Dial-a-Behavior-Protocols (DBPs) promise to modify the operation of a protocol on-the-fly with respect to syntactic and semantic packet contents and hide the complex implementation of network details from the programmer.

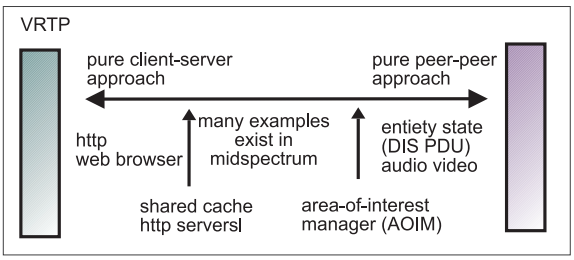

Figure 2.1: Spectrum of Connectivity as seen in D.Brutzman, VRTP

To give more detail about the available protocols and their advantages the most important Internet protocols are listed and explained below.

2.2.1 HTTP/FTP

Two of the earliest Internet protocols based on TCP/IP were HTTP (Hypertext Transfer Protocol) and FTP (File Transfer Protocol). Both are well established protocols for reliable data and file transfer over the Internet. While both protocols are TCP/IP based and establish a direct (reliable) connection between two hosts they do not scale with a growing number of users.

2.2.2 UDP Datagrams

UDP Datagram is a connectionless unreliable Internet protocol. UDP Datagrams do not establish a connection between sender and receiver as TCP/IP based protocols do. Since each UDP Datagram may take an individual route, UDP messages are sent connectionless in a best effort manner. Neither the order of

Information Type	Protocol (ISO/OSI Layer 4-8)
Files	HTTP,FTP,TCP
Events	UDP
Messages	TCP
Streaming Data	RTP,UDP

Table 2.1: Protocols used in Virtual Environments

packages nor their delivery is guaranteed at all. UDP Datagrams can be transferred by Unicast (peer-to-peer) or IP Multicast. When specifying a Unicast address a point-to-point connection is established, when using a Multicast address an IP-Multicast protocol is used for transmission. M-Bone (Multicast-Backbone), the experimental subnet of the Internet is based on Multicast UDP. Hosts can subscribe or ignore a Multicast packet at hardware (alternatively software) level by informing the network adapter (or driver) which Multicast addresses to monitor. This way, high-bandwidth streams can reach a large group of hosts identified by a single Multicast address. UDP Multicast supports grouping without changing the protocol functions. Grouping can not be used with the connection based TCP/IP protocol where connection establishing functions have to be changed. UDP is a very simple protocol. It realizes a few protocol functions such as providing a port number. With an IP address and a port number a sender or receiver is uniquely identified in the network.

2.2.3 RTP

The Real Time Transport Protocol (RTP) [Sch96] provides a framework to transfer real-time media data over Multicast or Unicast networks. The main function of RTP is to do application data typing and framing. This is achieved by specifying a packet format which includes identification of different media data via payload types, a sequence number and a timestamp. Specific payload types can be defined by applications for interpretation of specific media formats (payload formats).

RTP has a companion, the Real Time Transport Control Protocol (RTCP) whose purpose is to provide functions to encapsulate information about the participants in a session and to monitor the quality of service for the data conveyed by RTP. RTP/RTCP are now widely used in real-time audio and video conferencing applications on the M-Bone.

The data traffic of RTP and RTCP are sent in a best-effort way, so that it is not guaranteed that they will all arrive at the receivers. For continuous media applications it is most important to guarantee the end-to-end transmission delay while some packet loss is tolerable. RTP is suitable for these applications, because of its low-overhead, best-effort characteristics.

2.2.4 Multicast

Multicast is a crucial capability to scale up network capabilities. Multicast packets have class D Internet Protocol (IP) addresses which permits individual packets to be routed to multiple recipients without duplication on individual LANs [Dee89]. Host machines must intentionally subscribe to a Multicast address so that incoming packets to passed to the application. Thus Multicast packets have particular strengths in minimising bandwidth usage and minimising processor cycles. Since Multicast currently uses

only the User Datagram Protocol (UDP) and not the Transport Control Protocol (TCP) of the IP suite, Multicast streams are a connectionless and thus "unreliable" service. Lost M-Bone packets stay lost. No setup is required, no acknowledgments are used and no guarantee of delivery exists for this type of "best effort" service. The Multicast packets have a TTL (time to live) field which is used similar to the TTL field of IP messages but interpreted differently. Typical TTL values for Multicast packets are:

- 1, local subnet
- 16, LAN of the organization / institute
- 24, regional networks
- 32, Germany
- 48, Europe
- 64, worldwide

Multicasting is a good concept for most real-time information streams (such as audio, video and behaviors) since it avoids delivery bottlenecks and unwanted overhead. Numerous researchers are experimenting with "reliable Multicast" transport protocols which try to achieve a balance between reliability and potential congestion, typically by gaining occasional retransmission benefits without unacceptable acknowledgment overheads [Cro96]. The Multicast- Backbone (M-Bone) is one of the Internet's most interesting capabilities since it is used for distribution of live audio, video and other packets on a global scale. In other words, the M-Bone interconnects the Multicast capabilities of LANs across the Internet. M-Bone is a virtual network, because it shares the same physical media as the Internet and comprises a network of routers (M-Routers) that support global Multicast. Furthermore, it is possible to partition the LSVE (large scale virtual environment) communication space to exploit virtual reality by assigning different communication channels to different Multicast addresses, typically corresponding to geographic, temporal or functional areas of interest (AOI) [Mac95]. These considerations for reducing bandwidth and processor cycles while achieving global connectivity, are critical when scaling to arbitrary large numbers of simultaneously interacting users on the network. Since M-Bone is experimental, not all Internet users can participate. Figure 2.2 gives an overview of the Multicast network in Germany. Different tools are used to span the M-Bone. Besides routers, which have the special ability to transport Multicast packets and are the default hardware to build up the M-Bone, subnets which are not directly connected could be connected via M-Bone "tunnels". M-Bone "tunnels" are Multicast-Packets transported via the *IP in IP* protocol from a computer located in the M-Bone to the subnet not directly connected to the M-Bone. These packets are then distributed in the (formerly unconnected) subnet. The subnet has to assure that these Multicast packets are not spread out of the subnet again, which is normally achieved by using a router.

2.2.5 DIS

For the past seven years a major focus in networked virtual environments has been the Department of Defense's Distributed Interactive Simulation (DIS) standard, also known as IEEE 1278. The DIS standard for distribution (IEEE 95) describes the packet format for 27 protocol data units (PDUs) specified for use in military combat simulations. Only a few of these have general applicability. We are most interested in the widely used Entity State PDU (ESPDU). Entity state in this context includes linear and rotational values for posture, velocity

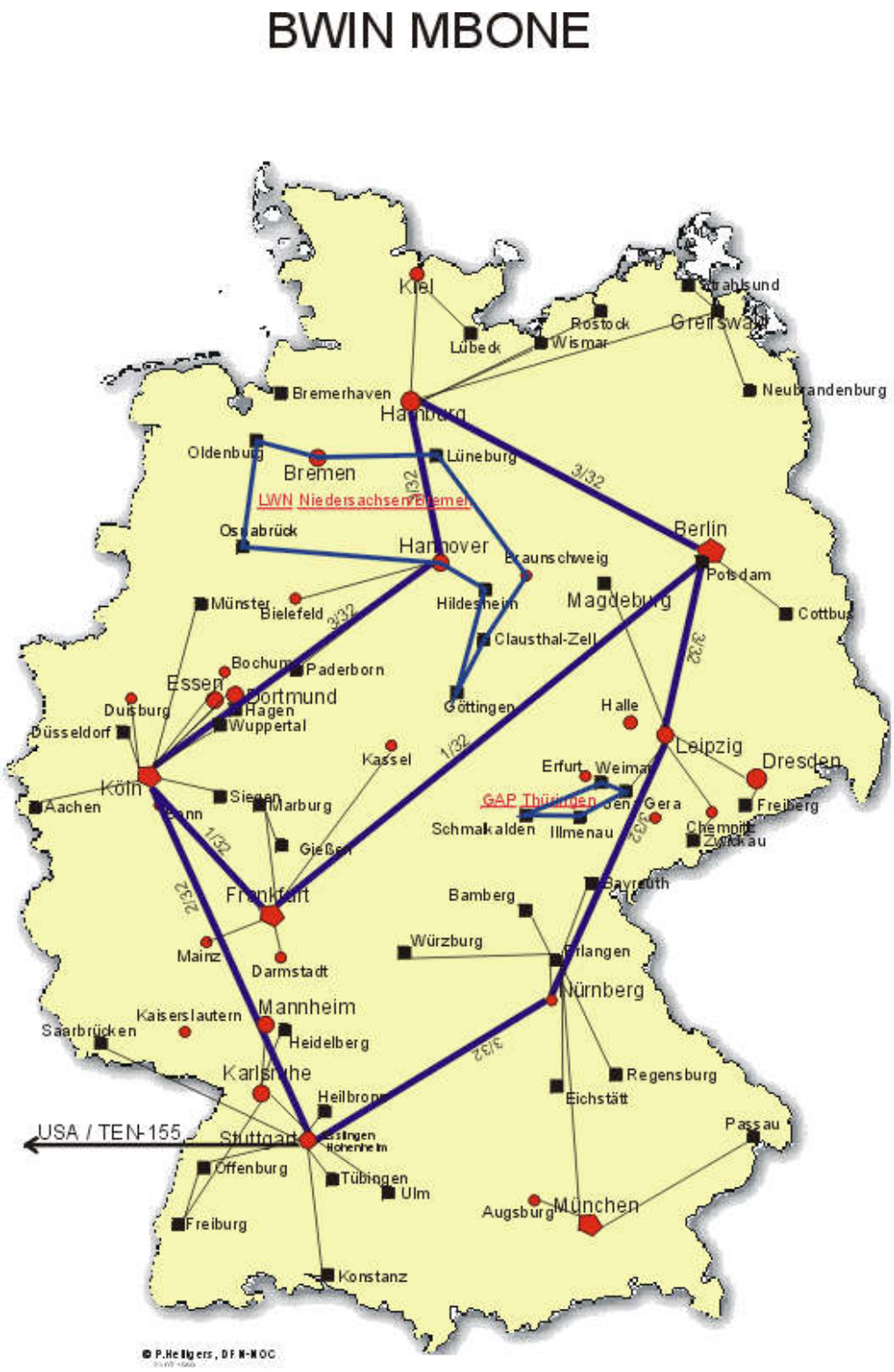

Figure 2.2: M-Bone, Map of Germany: Thick Lines Show the Multicast-Backbone, Thin Lines the Leaf Links Connected with M-Tunnels

and acceleration in combination with efficiently designed algorithms for dead reckoning and track smoothing 2.1.

The ESPDU is well suited for relaying physically based model information to numerous interacting participants in real-time. Meanwhile the DIS specification has been frozen and DoD (Department of Defense) development work redirected toward a hybrid distributed client-server object broker model, the High-Level Architecture/Run-Time Infrastructure (HLA/RTI). The IEEE DIS protocol is robust and effective in multi-user environments and can be suitable for game scenarios.

Further research is now going on to find a more general solution to extend the advantageous capabilities of DIS ESPDUs using a "dial-a-behavior protocol" approach. The current design includes an abstract PDU format grammar and Java applets that are able to switch parsed formats on the fly.

The hypertext transfer protocol (HTTP) [BLFF96] used for most web interactions was first evolved by efficiently combining capabilities of precursor protocols (ftp and gopher) and then optimising performance on client and server machines. Many other capabilities have since been added but fundamentally HTTP, provides a purely client-server relationship: a user can push on a Web resource and get a response, but there are no mechanisms for information sources to independently push back at candidate receivers. Inadequate support is available for light-weight interactions and real-time streams. The limitations of HTTP are widely known, but current HTTP next-generation (HTTP-NG) efforts appear focused on optimising and incrementally extending existing functionality [Gro96]. Thus even next-generation HTTP appears unlikely to support the Large Scale Virtual Environments (LSVE) networking requirements identified here.

The DIS Protocol is used by SimNet and NPSNet Projects at the Naval Postgraduate School in Monterey. NPSNet realizes an Area of Interest Management (AIM) by Multicast groups.

Problems of DIS are the availability of an entity state PDU only. To transmit information like a complex scene graph, PDU Packets are not enough, because only changes of object states could be transmitted. For a complex architecture a lot of other message types are needed, like the generation or deletion of nodes. DIS does not use a reliable transmission protocol, so lost packets stay lost, which may be acceptable for character movement in an LSVE but not if a reliable transmission of a changing object states is required. Another disadvantage is that the DIS concept transfers the state of each object frequently over the network, allowing participants to temporary disconnect, which of course puts a permanent load on the network, even if objects keep unchanged.

2.2.6 VRTP

Newest developments in protocols like VRTP [BZWM97] argue that large scale virtual environments need different data transport mechanisms encapsulated in one protocol. Brutzman et al. identified the need to have reliable and unreliable transportation facilities to generate multi-user virtual environments. They distinguish between four types of information necessary to create VE's: "light weight" interactions, as one information type is a message composed of state, event and control information as used in DIS Entity State PDU'S or other behavior reports. A complete message is included in a single packet without fragmentation. Light-weight interactions are received completely or not at all. "Network pointers" as a second information type, which introduces resource references to the network. Network Pointers contain

a reference to an object which could be accessed by the participants. The third information type is named "heavy weight objects". These are large data objects requiring reliable connection oriented transmission and are typically provided as a web query response to a network pointer request. "Real time streams", the fourth type of information, is live video and audio, DIS behavior, sequential graphic images or other continuous stream traffic that requires real-time delivery, sequencing and synchronisation, typically implemented using Multicast channels.

VRTP was developed to support both, client-server and peer-to-peer approaches of delivering network messages. This was motivated by recent work of the NCSA (National Center for Supercomputer Applications), which showed that intermediate functionality may be essential for scaling up.

Though supporting two different types of data transport (reliable and unreliable), the proposed "in between" solution is not described in the paper. An implementation of VRTP is not available yet. Prototypical implementations are expected second quarter of 1999, available in Java only.

2.2.7 LRMP, Lightweight Reliable Multicast Protocol

The distribution of reliable information on the M-Bone is very different from real-time audio and video conferencing applications. Since Multicast packets are sent as datagrams using connectionless protocols such as UDP, they may be lost due to network congestion and not delivered in order. While audio and video conferencing applications can tolerate some packet loss, the distribution of reliable documents should definitively implement a mechanism to repair the lost data blocks. Some Multicast Web tools use a gross grained scheme, the repeated transmission scheme, which is costly and introduces a large latency, thus not suitable for real-time applications. Packet loss repair should be handled in a fine grained fashion, i.e., at the packet level using a reliable transport protocol, much like TCP in the case of Unicast connection. Indeed, packet loss repair is performed at the cost of additional transmission delay which is admissible for the distribution of Web documents, but generally not for real-time audio/video applications. LRMP [Tie97] offers end-to-end reliable and ordered data delivery service to applications. Preliminary tests have shown that this protocol meets the requirements of file distribution over the M-Bone. LRMP does not provide a data history to support users which joined a session after some changes have been made. The main idea of LRMP is to provide repair information with every packet sent.

LRMP was designed as a general-purpose reliable transport protocol based on unreliable underlying network transport protocols such as UDP. It offers three important features: loss repair, ordered packet delivery, and adapted rate-based flow control. It is a complement to RTP. Initially, LRMP was intended to be implemented on top of RTP, but it was realized that there would be much redundant information carried in LRMP packets and RTP packets, so it is wiser to implement LRMP as an extension to RTP/RTCP. Five control packet types are specified in LRMP: ECHO and ECHO_ACK packets are used to measure the round trip time between two users; NACK packets are used to report packet loss; SYNC packets inform the receivers the of the outgoing sequence number; SYNC_ERR packets report an unrecoverable sequence error at a receiver. All these packets are implemented as RTCP control packets. Like RTCP, several LRMP packets can be multiplexed into one outgoing packet. In LRMP, application data are just sent as standard RTP data packets. It is the application's

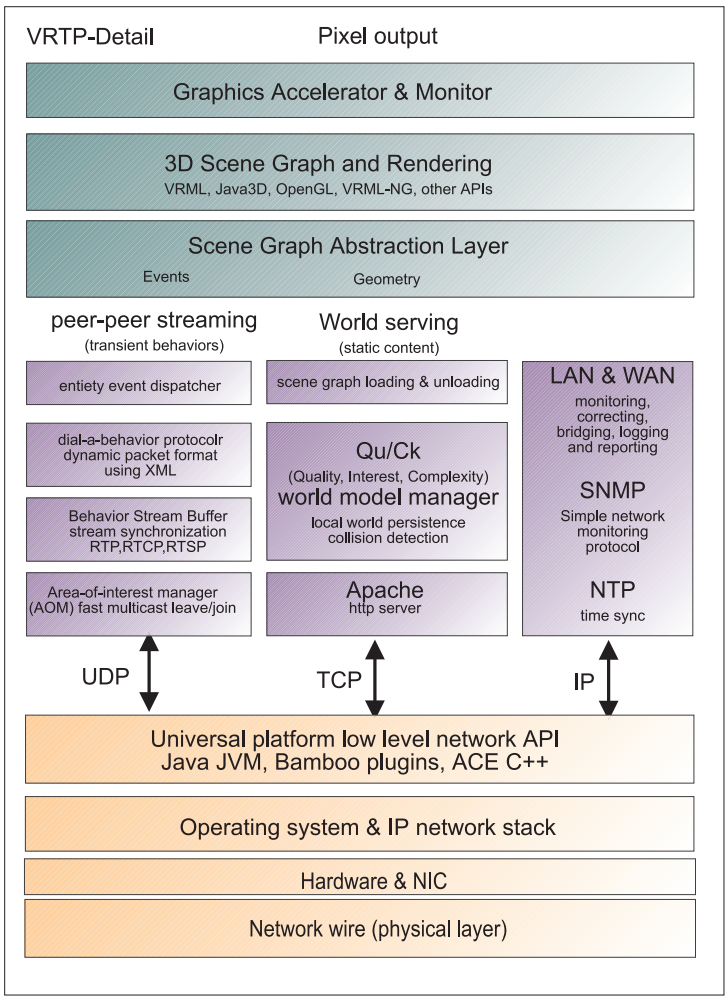

Figure 2.3: Design of VRTP

responsibility to use a specific payload format to encapsulate media data and fill the timestamp field. LRMP packets and RTCP packets together with the sequence number carried in RTP packets provide a set of information sufficient to make a reliable transport protocol. LRMP was carefully designed to work against possible loss of control packets.

A receiver detects packet loss either by checking if there is a gap between the sequence numbers of two successive packets or by identifying the sequence number given by a SYNC packet does not correspond to the last received packet.

A NACK packet in LRMP includes information about the first sequence number lost (16 bits), a bit-mask (16 bits) for succeeding lost packets. A LRMP receiver uses the random timer value for sending NACK's and keeps track of NACK's sent by other receivers to avoid NACK implosion.

Upon receipt of a NACK packet, repairs are sent immediately by the original sender if the request is considered not duplicated. LRMP treats resend requests of the same packet as duplicated if they are received within 0.5 seconds. The sender gives higher priority to sending repair

packets than sending normal packets.

While in-order packets are immediately delivered to the application, out-of-order packets are maintained in the cache by LRMP at receivers in waiting repair packets. Once missing packets are received, cached packets are removed from the cache and delivered to the application.

The reception cache has a limited size, but should be large enough for loss repair. However, if there are many out-of-order packets and repairs do not arrive, the reception cache may become full. In this case, there are two choices, either to drop new packets or to clear old packets to keep synchronized with the sending process.

Unlike TCP which uses a window based flow control, LRMP uses a rate based flow control mechanism. That is, LRMP sends data packets, including repair packets, at a rate specified by an application. This rate is adapted dynamically to better suit available network bandwidth.

The algorithm currently used in LRMP is based on loss statistics collected through RTCP RR's (receiver reports). Only "bad" receivers affect the flow control. Transmission rate is decreased when there are many receivers with a significant loss rate and increased when there are a few. The purpose of this algorithm is to keep an acceptable speed for the majority of receivers and to try to satisfy a few worst case receivers when possible.

While 100% reliability is attractive, it is difficult to achieve in practice. To ensure 100% reliability we have either to use a flow control mechanism (like in TCP) which is adapted to the worst receiver or, in absence of this adaptation, to keep very old data for any possible retransmission. The first solution is excluded in LRMP. For the second one there are two ways to realize it, either old data is kept in the cache of the protocol entity or by the application. In case of continuous data stream, keeping all data by the protocol requires a cache with infinite capacity.

If, due to network partition or a receiver with a bad link, reception failure occurs, i.e. a receiver finds a too large difference between the currently received sequence number and that of the first lost packet, it will send a SYNC_ERR packet (upon timeout) to report a serious synchronisation error. It also cleans its cache and tries to synchronize with the current sequence number. At the sender's side, on receipt of a SYNC_ERR packet, LRMP will notify the application of such an event and of the interval of lost sequence numbers. It is the application to decide to take any remedy actions.

For example, it may ignore the event for continuous data stream or re-send this part of data according to the level of importance. This mechanism allows applications to control the level of quality-of-service and thus prevents the disturbance caused by a particular receiver with a very bad network connection.

2.2.8 DWTP

The Distributed World Transfer and Communication Protocol DWTP [Bro97, BS98] is an application layer network protocol for shared virtual environments and based on top of TCP/IP and UDP/IP allowing a virtual environment to transmit and receive several types of data:

- events
- messages
- files
- streams

DWTP uses events to keep copies of a distributed shared environment consistent where the quality of the connection could be specified and messages are a set of predefined events. Files require a reliable peer-to-peer transmission, and streams are used to transmit a continuous flow of data. The network

structure of DWTP is based on participants and daemons (server), which implement different parts of the protocol. Reliability daemons detect transmission failures, recovery daemons provide Unicast connections or recovering of lost packages, world daemons transmit the virtual world contents to new participants and Unicast daemons realize a Unicast connection for participants without Multicast capabilities.

Broll [Bro97] identifies five types of data to be transmitted via the underlying protocol

- reliable peer-to-peer transfers of (large) files
- reliable and unreliable transmission of (small) events
- reliable transfer of (medium-sized) files to a group
- unreliable transfer of stream data to a group
- unreliable peer-to-peer transfer (optional)

The reliability daemon is used to detect transmission failures. It sends a positive acknowledge message when receiving packages (ACK's). It avoids negative acknowledge messages (NACK's) used in [WMK96] in order to prevent the NACK explosion effect. This effect occurs when a large number of recipients is separated from the network. In this case, all recipients will send a NACK and lead to a partial congestion of the network. Broll states that network splitting is one important reason for transmission failures, and the mechanism of NACK transmission would intensify the congestion problem.

DWTP provides access to the Multicast data for non Multicast members by using Unicast daemons. The needed network address is provided by the world daemon at world access. Unicasting is completely transparent to the user of DWTP, and the Unicast participant connects the Unicast daemon using the encapsulated data of the world daemon. The daemon forwards the messages to the Multicast group as well as to all other Unicast participants connected to the daemon and distributes Multicast messages to the Unicast participants.

Broll argues that this environment seems to be rather complex, because there are at least three daemons needed for a distributed environment. A second problem is scalability, because world and reliability daemons can only handle up to 20 participants, especially when users have to be provided with streaming audio, or use the Unicast daemons when connecting via modem. So the world has to be split up to allow several daemons to serve all users, rising the problem of transmitting information between daemons. Transmission of data between daemons is not handled by DWTP, because groups may be separated by regions. Furthermore, the problem of users migrating from one group to another is not handled by DWTP.

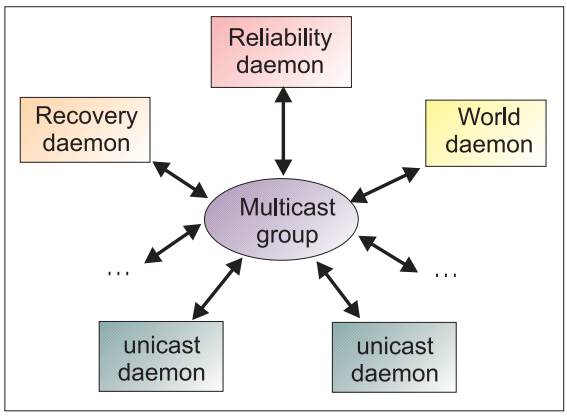

Figure 2.4: Design overview of DWTP

2.2.9 "Unreal" Networking Architecture

Because our problem is actually also a problem of game developers, we are well advised to

have a look at the Unreal Networking Architecture (UNA). In [Swe98] Sweeney describes the network engine of the game "Unreal". He describes the objective of a multi-user game as a shared reality in which all of the players feel to be in the same world, seeing the same events as the other participants from different viewpoints. In the "Unreal" approach a Client-Server model is used, based on TCP/IP and UDP/IP connections. The server executes the same game logic as the clients, allowing the server to predict participant movements. The proposed approach monitors available transmission bandwidth and apportions the amount of bandwidth in to separate message streams of diverse importance. Important messages are transmitted for sure, while unimportant messages are only transported if bandwidth is available. All messages to be sent are priorized, assuming that there is not enough bandwidth to transmit the entire state of the system. The relevant set of actors is determined. Rules could be defined to calculate this set of actors. These rules are based on geographic, static or dynamic issues, if an object belongs to somebody or if it is in the line of sight or was interacting with the participant a short time before. Whilst calculation of the relevant actors and transmission of all of their data could consume too much bandwidth, actors can be priorized by importance. So enemies and projectiles are important while decorative elements are considered to be irrelevant. The approach distinguishes different types of data to be transmitted:

- actor replication
- variable replication
- function call replication

While actor replication is based on the set of relevant actors, variables are always transmitted reliably. Function calls can be executed reliable and unreliable, depending on game logic. The system provides essential tools to determine necessary actions, and so it can decide which messages have to be sent and which can be omitted. To save bandwidth the "Unreal" engine transmits the values of vectors and rotators as 16-bit numbers whereas the values for "Pitch", "Yaw" and "Roll" are transmitted as bytes.

Further approaches were made to handle dead reckoning and time synchronising of the connected applications.

Though the server may be the bottleneck of the "Unreal" Engine the bandwidth approach has several advantages. At first a Multicast connection is not available for normal end customer participants, and because there is no variable infrastructure, servers could not be set up to convert Multicast to Unicast messages. Thus the use of a server for UDP messages may be a good design. Another advantage is the idea of bandwidth restrictions arising in the need to communicate to the participants via modem links.

2.2.10 Discussion

We can see from the description of the presented architectures that distributed multi-user systems are still an ongoing field of research. Some of the presented protocols are not well suited for our kind of problem. So DIS was developed to meet the demands of a distributed warfare simulation, and it offers only one PDU suitable for non military applications. Beyond this, it does not provide mechanims to transmit scene data and congests the network with a ground load of repetitive messages. Broll's DWTP seems to be too difficult to handle because it uses too many servers and does not implement any mechanism to handle multi-server setups. Nevertheless, his idea of using Unicast daemons for modem connections is interesting. VRTP is under development and not yet available. Latest developments of VRTP will influence the VRML-Java-DIS discussion group and surely will become an

important protocol in the future. Gaming systems like the "Unreal" engine are very close to our application but restricted to non-Multicast protocols and may not scale well with a greater number of participants. Other reliable Multicast protocols are too application specific or produce too much overload or do not guarantee the delivery of messages. Common to many implementations is that they separate messages by importance and try to use different transport methods depending on the message type.

2.3 Data Transport in distributed Environments

When transporting data to a group of session members in a heterogenous network environment many kinds of errors can occur. Depending on the underlying protocol, simple failures like transmission gaps and packet ordering could be solved. When using Multicast the underlying UDP transport does not provide this feature and a protocol like RTP may be used to solve packet ordering. As shown in the previous sections a suitable Multicast protocol for secure packet delivery is not available by now. Beside packet ordering and secure packet transmission, other problems of importance in a distributed multi-user environment are:

- during a session a participant may be disconnected for a shorter or longer term. A short term disconnection may result in a retransmission of the last data packets which might have been missed. A long term disconnection may result in a complete disconnection from the session.

- The group may be divided into small subgroups, splitting an experimenter from the audience.

- A participating server may fail, and message transport may be delayed.

- A new server has to be chosen.

- When users join the session, they should receive the actual representation of the database and not a copy of the start database.

- When users leave the session, their data, like avatars or simulative components, should be removed from the actual common database.

Type Checking

Network protocols just transmit byte packets. Implementing data types and type checking enhances the usability of the system for the programmer. Systems like Corba (see Section 2.4) keep the implementation of type conversion transparent to the user and make common data types available for heterogeneous platforms. If the transport protocol does not support data typing, these mechanisms have to be implemented by the application programmer. For example, due to a lack of appropriate mechanisms, we had to implement type conversion for all internal data types when using a Multicast connection.

Marshaling

When using different platforms the data types may not be compatible. Different internal representations may be used to store a number. A heterogeneous system should ensure that data does not lose any accuracy in representation over transport between sender and receiver.

Byte Ordering

In heterogeneous environments the byte ordering (little or big endian) may be different. The

transport mechanism should care for these differences and ensure correct byte ordering. Of course this mechanism should be transparent to the application programmer.

2.4 Corba

Corba, the Common Object Request Broker Architecture [Gro94], was brought to life in 1989 by the Companies 3Com, American Airlines, Canon, Data General, HP, Phillips, Sun and Unisys, which formed the Object Management Group (OMG). This chapter describes the Object Management Architecture (OMA) of which the Common Object Request Broker is a part.

A central part of the OMA is the Object Request Broker ORB. The ORB provides transparent information exchange between client and server objects independent of the current platform. The ORB provides Object services to realize Corba applications. These object services can be combined and allow complex applications. One important service is to find all other available objects by name or by type.

Corba is a standard defined by the OMG in the OMA reference implementation. It allows the user to concentrate on the development of object features instead of thinking of network problems. Corba provides an interface user application developers where they are not aware of specific object aspects like:

- *location:* the application programmer is not informed about where a special object is located, especially where the code is actually executed.
- *implementation:* the application programmer may not know how the object is implemented.
- *parameter conversion:* the parameters which are necessary to call an objects's method are packed. The application programmer has not to care about byte ordering.
- *execution status:* it is not necessary to know wether an object is ready to receive data at all. Eventually, the ORB starts a new object transparently.
- *transport mechanism:* the application programmer does not know how the parameters are passed to the object.

Communication with a Corba Object is established using an object reference. Object references are provided when a client creates a distributed object. A distributed object is created by calling an object factory. Before calling an object's method a client has to know the object's interface. The object's interface is specified in the IDL, the Interface Definition Language. The IDL describes the interface to the methods the distributed object provides. The IDL specification is platform independent and supports some common data types such as long, double, boolean, structs, unions, sequence and string. When compiling an IDL the application programmer gets a proxy class called *stub*, which passes function calls transparently to the Corba server. The *skeleton* is the counterpart of the stub at the server-side which implements the functionality provided by the object.

2.5 Discussion

Comparing the described Multicast transport protocols some of the described network problems stay unresolved: marshalling, type conversion and transmission of complex objects.

Using a Corba object for data transport frees the application programmer from the discussed network problems and ensures a reliable 1:n

transmission of the object messages and data. In our implementation we use the free ORB implementation ORBACUS [Lau] to implement a reliable transport layer for the distributed application MRT-VR.

Chapter 3

Design and Implementation of MRT-VR

In the previous chapter we presented different transport protocols to pass messages over Multicast connections. Many of the presented protocols use a secure transport mechanism when the data transport to a Multicast participant fails. Some of the protocols have major drawbacks when used with our application or are not available at all. Comparing the protocols and the transmission errors they cover it reveals even more problems. In the Section 2.4 we described the use of Corba for secure data transport.

We now motivate the design of MRT-VR based on several important specifications, which led to the current implementation.

The design of MRT-VR consists of three main parts. We distinguish between data-transport, data-replication and the application with the visualization components. Data transport realizes the transportation of messages at the network level by different protocols between instances of MRT-VR. Data replication assures the consistency of scene databases, while the visualization assures a fast, multi-platform, multi-quality display of the virtual environment.

For the implementation we used multi-threading for the data transport and data-replication mechanisms to increase the system performance and availability. During a session the user may interact with the system and the images have to be recalculated and re-rendered. The system has to be highly interactive and should not be blocked by data transport.

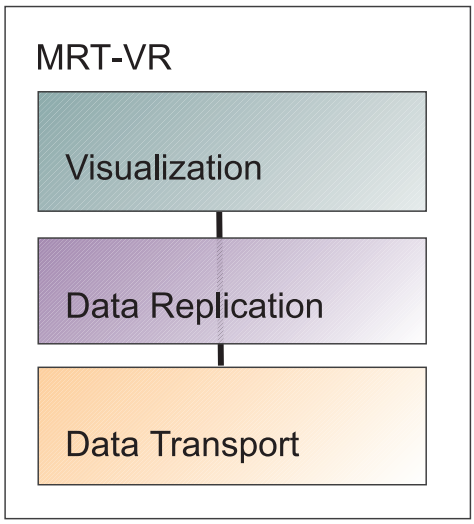

Figure 3.1: Design of MRT-VR

Data transport between connected users can be implemented in a synchronous, a semi-synchronous or an asynchronous fashion. Using a synchronous data transport the application is blocked until a remote service has terminated. Using a semi-synchronous communication scheme the application is interrupted when receiving data packets while a polling mechanism is implemented when data packets are transported asynchronously. MRT-VR should use an asynchronous data transport mechanism. The application has to determine the best time for a scene update.

Because the system should be platform inde-

pendent, the data transport has to handle the byte ordering (little and big endian) and the conversion of different binary formats. Type checking should be implemented if not supported by the underlying transport mechanism.

As a major feature compared to other architectures the system transports objects in their full semantic representation over the network and not as a set of triangles approximating the shape of the objects. Of course, the transport of triangles is supported by an object called indexed face sets. The transmission of construction parameters for objects does allow to generate any type of object remotely. This does not include only classical algebraic scene objects like spheres or cones. Any other object implementing a special function like controlling a behavior or an edit function can be added to the system and transported by sending its construction parameters.

Transporting of objects as semantic entities has some major benefits:

- the saving of transmission bandwidth by transporting a center-point and a radius to generate a sphere at the remote site instead of transporting a fairly large set of triangles.
- the ability of the object to generate *any* quality of representation needed for the visualization.
- the independent calculation of intersection tests from the object triangulation.

This approach keeps the knowledge of an object and its properties, for example assuring that intersection tests are independent of the image resolution, allowing to render a raytraced image from the transmitted objects at arbitrary quality, which is not possible by any other system. This mechanism will preserve object properties even after transmission. The remote application can decide in which quality a sphere may be approximated for triangle based rendering. Adjusting the approximation quality without losing any information allows the remote renderer to adjust the approximation quality according to the underlying hardware.

3.1 Implementing a Distributed VRML System

3.1.1 Intention

MRT-VR was developed to support distributed learning and discussion. Through the use of VRML, MRT-VR is open to any 3D-application. MRT-VR can be used to explain architectural-models as well as molecular design. MRT-VR distributes scene changes to all participants, supports even unexperienced users while moving in 3D-space, shows other users by selectable avatars, and allows, in a teaching environment, to assure that all participants see what a particular user (e.g. the teacher) wants them to see. An important issue on supporting people in teaching and collaboration is the easy handling of the employed software. So we took some effort to support the user with easy program access, movement in 3D-space and a simple way to interact with the program.

While using different transmission protocols, MRT-VR can be used to meet several application environments. Through its support of OpenGL [Sil93] and XGL [Sun93] in combination with the use of the MRT-Binary-Protocol (MBP) external simulations can process their data to MRT-VR and use it as a specialized display server.

In combination with other M-Bone-Tools such as VIC (video [MJ95]), VAT (audio [JMa]) or Whiteboard (paper [JMb]) or communication software like NetMeeting [Cor], MRT-VR provides distributed interaction with 3D

content (VRML or MSD [1] scenes MRT scene description). Figure 3.11 shows the integration of MRT-VR in the SDR-Tool [Han96], which allows an easy use of MRT-VR in combination with video-, audio-, and whiteboard-tools.

The dynamic aspect of MRT-VR is not restricted to the visualization of other participants through avatars or selection of objects (picking). MRT-VR allows a complete modification of the entire scene description including avatars, camera positions and the state of the MRT-VR viewer (view modes, camera positions, etc.) through external programs or other instances of MRT-VR. The design of MRT-VR allows an easy modification or enhancement of the supported command set as well as transport mechanisms.

3.1.2 Scenario

Referring to the 'Big Picture' again an experimenter is operating a robot platform at a remote location. The robot's motion has to be controlled remotely by the experimenter. Because high speed Internet links (needed for continuous live video) are not always available. Controlling the robot could be a difficult if not impossible task, because the feedback of the robot's motion may take too long. We could help the experimenter by using a visualization of a virtual environment:

The robot and the robot's environment is displayed by MRT-VR using a VRML-Model. Employing the MBP (MRT-Binary-Protocol [HF98]) position changes of the robot and of the environment are processed via the Internet using significantly less bandwidth than a video-stream.

[1] MSD, MRT Scene Description, is the native file format for MRT-library scene description language

3.1.3 Access Rights

As we stated earlier, we need different user types for different scenarios. So, during a presentation it may not be necessary to have avatars of other participants or even to allow the modification of objects in a scene. In a collaborative environment, however, we might want other users to be represented by avatars, and we might also want to support the insertion and modification of objects into the scene. During a discussion we might need 3D pointing devices inserted in the scene and allow highlighting, but we might not want that the other participants modify the objects in a scene. These examples indicate the need for a locking mechanism and the use of access rights.

In a central server approach access rights could be managed by the central server and controlled by the session initiator. In a distributed environment the session initiator has to manage the object rights. Saar [Saa98] describes object rights management implementation similar to a Unix files system. In our implementation we use a comparative solution with the flags named R_FULL allowing to modify every object, R_NONE, the user may not even have a look at the scene or R_READONLY allows to read the scene. Existing objects can be changed if right R_CHANGE is granted. Access rights are granted and may be changed during a session by the session initiator only.

3.1.4 Interaction

To support unexperienced users, MRT-VR has some built-in functions to avoid awkward situations. Through the use of the MRT-Library [Fel96] a 3D graphics library developed at the University of Bonn, ray-casting operations, such as intersection-tests with scene objects, can be calculated very efficient. MRT-VR uses intersection tests to create an intuitive movement

within a scene, while exploring it. These enhancements are described in Section 3.4.

3.2 Distributed Scene Database

When developing a distributed multi-user environment several aspects have to be taken into account. A multi-user system should provide shared 3D-views for different users. The users should be able to share a common point of view or have different viewpoints. The system should provide functions for distributed scene editing, so that distributed or CSCW is possible. We call this situation a "conference" or collaborative "session".

A collaborative session expands a common 3D-scene viewer by

- management of participants
- management of cameras and viewpoints
- data replication
- management of access rights

As stated in Section 3, it is remarkable that most of these problems are similar to problems that arise in distributed data management systems as described in Section 2.4 when comparing the scene graph to a distributed database. In our research we implemented several versions of the MRT-VR based on unreliable Multicast transport and implemented the necessary database functions. While using these systems many problems arose and led to more complex environments. As stated in earlier chapters, reliable Multicast protocols are currently not available or have major drawbacks when applied to our application.

Some of the presented protocols like DWTP use Unicasting and special servers when a secure transmission is needed, reducing the Multicast transport to a common client/server model. This led to the idea of using Corba or DCOM [Cor96a] for the distributed scene database. In [Leo99] we compared actual distributed databases and implemented a version of MRT-VR completely based on Corba.

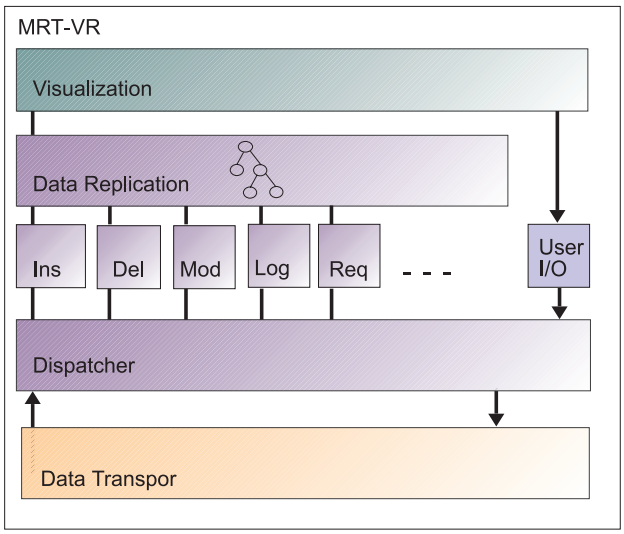

Figure 3.2: Data Replication

For fast access to the scene data, the scene graph has to be fully replicated at every participant.

A native implementation of a Corba system would use a derivation of Corba objects and MRT objects for the scene graph allowing a direct synchronisation of connected participants. The need of a fast scene graph traversal does not allow a direct coupling of Corba and MRT, because the synchronisation through the Corba ORB may lead to a synchronisation of the Corba object with the other distributed objects via the Network, and would take too much time for a real-time visualization.

By de-coupling MRT and Corba with a scene management we achieved the independence needed for real-time rendering. To have a complete redundant scene graph at every participant is an important feature when the

actual server may fail. Besides, it allows every other participant to become the new server, which leads to a very stable system design.

To minimize scene change synchronisation over the network we implemented the "primary-copy" [Sto79] mechanism in the Corba transport. Using the primary-copy mechanism the scene changes have to be written into a write buffer and passed to the server. The server passes these changes to the other participants asynchronously from the write buffer to their read buffer, leading to a delayed delivery of scene changes. The scene is updated when the current participant has some idle time or when a timer has run out (e.g. 10 times a second).

As stated before, we distinguish different messages by importance. We use Multicasting and Corba concurrently avoiding the overhead of implementing a distributed database system for secure transmission and using the performance of Multicasting for messages which are inherently of Multicast type, such as camera data and position information or chatting text.

In the next chapters we describe the session management in more detail and give information which part of the library contains the responsible code.

3.2.1 Participant and Viewpoint Management

The MRT-VR participant management is separated in two parts, the first part concerns the Corba communication, the second the Multicast communication.

Some of the important session management functions are:

- start and stop of a session
- login and logout of a user
- participant list
- change of user information (name, info, usw.)
- management of access rights
- support of 'machine' type programs for simulation
- selection and loading of a user representation (avatar)
- transmission of camera (avatar) position

All users currently logged-in are held in a public user list. This user list contains all information that the user enters at login, such as name, e-mail address and comments. Some of this information, such as comment and e-mail address, can be changed or updated during a session. Some are excluded from updating, e.g. the user's nickname, because they are used to identify a user.

The Corba server assigns a unique user identifier to each user. This user ID is used in the MUI (Multi-User Interface) library to identify a user, and is commonly referred to as UID. The public user list holds the cameras which are updated by messages from the participant's change of viewpoint. So the user and camera management are strongly coupled, which is reasonable, because each logged user has its own viewpoint.

The public user list is commonly accessible by the Corba and the Multicast session and is to be used to display the user's information in a system dialogue.

Each session implementation has a second own private user list to store implementation dependent information about users. The Corba implementation stores information about user rights and timing, whereas the Multicast user list stores information on Multicast user IDs and statistics.

The user lists and their corresponding message passing functions are realized as independent threads. These threads handle sending and

receiving of user data and camera messages. An application program like MRT-VR accesses user data through the session API t_ConfInterface and the public user list t_PublicUserList, which provides all necessary interface functions. User disconnections are handled by the user lists and are transparent to the programmer[2].

The method setCamera() of the class t_ConfInterface sets the actual camera position for a participant. Camera data is transported only if it has changed since the last call of setCamera(). The cameras of the other participants can be found using the methods getCameraIDs() and getCamera(). The method getCameraIDs() returns a list of UID's for all users currently participating whereas the method getCamera() returns the camera of a user by passing its ID. If a user has registered an avatar, the position of the avatar is automatically updated by the user list when a camera change message is received. It is necessary to change the avatar positions at every remote site by the user list to avoid unnecessary object change messages: If the change of a camera position resulted in a change of avatar and this change of geometry was registered, all participants would receive an update message for the geometry. This change in the scene structure would be transmitted to all connected participants, causing a constant load to the Corba server when camera changes occur. To avoid this, every participant updates scene avatars from camera positions.

Avatars can be registered for every session participant with the communication rights *R_FULL*. Any 'scene' can be used to represent a user. A user has to construct or load a scene which is referenced by a reference object (see Section 4.1.4). The use of reference objects for avatars allows an easy transformation of the avatar's position. By the use of the visibility flag, an avatar can be selected from a set of avatars allowing to express emotions or machine states when an avatar is issued with an external simulation. Therefore we use a sub-scene node for all avatars. Since every avatar has to be moved, the whole scene can be moved by one reference object. Using an additional reference object for each avatar the corresponding "visibility" flag can control the appearance of the avatar (See Figure 3.3 for details).

Beside real persons it should be possible to provide access to scene and camera data to some simulative or control programs. These programs are used to push simulative content to the scene or to connect external objects, such as robots to the scene. These programs may be called "machines" . Machines may gain scene access, but not necessarily have a viewpoint or a camera attached. Additionally, it may be necessary to grant special rights to these machines which allow to insert only "private" objects into a scene and do not allow to modify or delete existing elements.

These machines should have access to the scene data to calculate collision detection or use the state of an object for special calculations known from the VRML97 standard. For example, a proximity sensor may be realized by an external machine which has full scene access and could implement a special behavior when an object comes close. In contrast to VRML this behavior is re-usable for any other scene and directly visible to all participants. The use of machines gives the system almost unrestricted enhancement possibilities. We used machines to couple different simulations to the MRT-VR. See Chapter 7.2 for more information on coupling a simulation with the MRT-VR.

3.2.2 Data Replication

Scene management keeps the scene data consistent between the connected users. Scene man-

[2]The prefix t_ is used to indicate a class type definition.

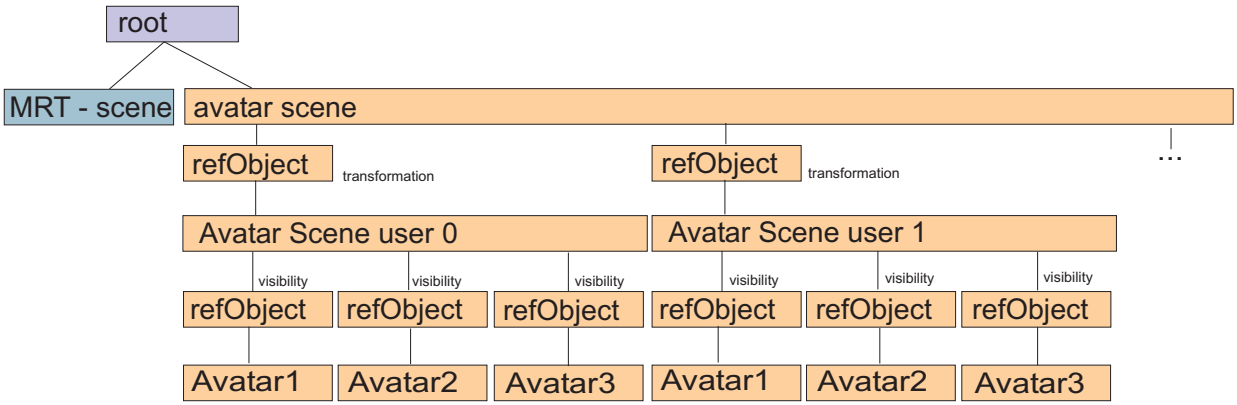

Figure 3.3: Embedding of Avatars into the Scene Structure

agement is built upon two independent parts. The first part is the scene graph API, which provides interfacing functions to insert, delete and modify objects in a scene. The second part has to handle the messages, coming from or sent to the Corba or Multicast transport to transmit appropriate change messages to all participants.

As the scene data is visualized by the MRT library, the MRT scene graph is used to render a scene. Objects contained in the MRT scene graph are described in Chapter 4.

The scene graph API was developed for the MRT library to extend the object-oriented design of MRT to a dynamic scene buffer called "object list". The object list is used to receive changes from outside and inside the application and to reconstruct a MRT scene graph on demand. The object list is a stand alone class t_ObjectList which can be easily added to the current MRT implementation to enhance the MRT library to a full scene editor, and provides functions to

- insert objects (light, object, scenes, shader)
- change position of an object in the scene hierarchy
- modificate object parameters, especial reference objects
- delete objects
- mark objects
- add sub-scenes (e.g. Avatars)

The scene hierarchy of MRT provides the class t_FullScene which contains all information necessary to render a complete image.

The class t_FullScene contains lights, the geometric scene objects, a camera list, object references and a shader list. The object-oriented design of MRT was originally developed for fast ray-tracing of geometric objects, thus, avoiding the use of transformations while traversing the entire scene graph. Base objects of the MRT library are constructed in world coordinates. For the approximative rendering of the objects a boundary representation (BREP) is constructed in world coordinates and not modified afterwards. This design is extremely good for fast ray-tracing applications and static scenes but not suitable for dynamic multi-user environments. However, the extensible design of MRT allowes the implementation of the object list to provide a dynamic interface. When constructing an object list from a given scene, changes may be applied to the scene structure, and afterwards these changes are passed back to the t_FullScene class. To access the object data, all

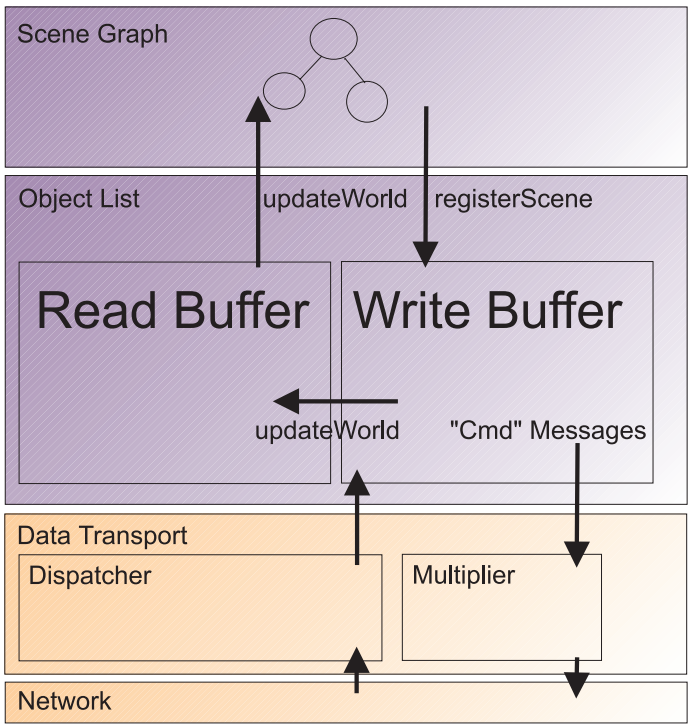

Figure 3.4: MRT-VR Data Replication with the Class t_ObjectList

objects were provided with a method writeObject() to return their construction data. Storing these construction parameters in a standard template library (STL) tree allows fast access of all object parameters in $O(logn)$ time. Each object can be retrieved by an object identifier (OID), which is the index in the STL map. The object OID is passed to an object when registering the object at the object list. Object references, which are used in scene containers, reference objects or as shader pointers, are resolved by their OID allowing to handle a complete MRT scene graph, including shaders, lights, objects and scenes. Single objects or the complete scene structure are registered when the scene is constructed and where the OIDs are assigned to the objects. An OID of value 0 indicates that a new and yet unregistered object has been created and inserted in the scene and thus, has to be registered.

Before a scene object can be changed by a participant the object has to be locked by the method lockElement() of class t_ConfInterface. This function may block the application when used with Corba, because a lock has to be requested by the server. A lock is requested only if the local lock list does not contain a lock for the specified object. The function returns true if a lock can be obtained by the server. If the object is locked it may be modified. After modification the object has to be unlocked using the unlockElement() method. The local lock list is a copy of the server's lock list working as a proxy for lock accesses.

When implementing an object editor or during a session it may be suitable to mark objects. Objects are marked by modifying their shader from their current color to a color indicating the marking. The method markElement() will highlight the specified object and exchange

the shader, which is reconstructed with the unmarkElement() method. The shader used for highlighting can be exchanged using the setMarkShader() method. This shader is not forwarded to the other participants so that any client could select an own mark shader. Objects can only be marked, if a lock is set, and will be automatically unmarked if the lock is freed. Locks and marks are stored in the user list, which results in a release of all locks and marks when a user leaves the session.

Objects and scenes are inserted in the object list database by the use of register commands. The methods registerLights(), registerShader(), registerScene() and registerObject() add elements to the scene database. When inserting a shader, a scene or an object the container (normally a scene) in which the object should be inserted has to be specified. If the container is not specified the object is inserted into the base scene. A registration is recursively done for all objects in a container, so that one registration call is sufficient for a whole scene.

When an object should be deleted, after a lock has been granted, the appropriate deleteElement(), deleteScene(), deleteLight() or deleteShader() method has to be used.

Object modification is done by calling the transformElement() method. This will change the transformation matrix of any object in the scene. Because every object can be reconstructed using a transformation matrix the objects position, scaling and rotation can be controlled by this matrix. The *relative* flag allows a matrix to be applied relatively or absolutely, where the objects matrix is calculated by T=T*S. If the *absolute* flag is false, the matrix T is replaced by matrix S. Using the translateElement() method allows the translation of any object in the scene, absolutely or relatively depending on the *absolute* flag, positioned by the value specified in the 3D vector.

The function exchangeShader() changes the shader of a given object, if the shader is not registered, this will be done automatically.

Object modifications can be bundled using the openTransaction() and closeTransaction() methods. All locks used for a transaction are requested at the beginning of the transaction and, if granted, all actions specified between openTransaction() and closeTransaction() are transmitted as one data packet. If not all locks can be granted the modifications are reset and all objects are unlocked and unmarked. The function abortTransaction() uses method reconstructWorld() to reset the changes. Transactions can be nested, but if a (sub-)transaction fails, the whole transaction will be aborted and abortTransaction() will perform a rollback.

All object changes are marked locally with a *Change_Flag* at the modified object. An object can be reconstructed on demand to apply the changes. To increase system performance every changed object is stored in a cache called "quick list", avoiding the complete traversal of the scene database to find all modified objects. All modifications can be applied by an updateWorld method of class t_ConfInterface when updating all objects referenced in the cache. This cache is faster than scanning all objects for changes if the length(ObjectList) is greater than log(length(ObjectList)*length(QuickList)), which is valid for most transactions and of course for modification of a single object.

Several flags indicate different object state changes which may be used to implement different update behaviors:

- *CHANGE_NONE:* no changes
- *CHANGE_DELETE:* object is deleted, container CHANGE_UPDATE
- *CHANGE_NEW:* attached to a new object, container CHANGE_UPDATE

- *CHANGE_UPDATE:* attached to changed objects, reconstructed or changed, re-creation is omitted if possible
- *CHANGE_TRANSF:* set for objects which can be changed without reconstruction ,e.g. reference objects

The second part necessary for scene consistency is to keep the object and user lists consistent for all users. For each session that the application programmer initializes, he has to select a transport mechanism. In our application MRT-VR a Multicast and a Corba session are started at the same time. The Corba session uses the Corba ORB for data transmission, while the Multicast session uses the Multicast RTP protocol to exchange messages.

Both session implementations provide the same interface to register scene changes which have to be forwarded to the other participants. Using the Corba implementation a reliable transport is chosen. While using the RTP protocol the messages are transported using Multicast and may not reach all participants.

Transmitting the object list data with Corba requires Corba compliant data types specified by the OMG IDL. Convert methods of class t_Convert map MRT data types to Corba types and vice versa. The class t_Convert converts the parameter list of MRT parameter in the object list to a parameter list of Corba compliant types which can directly be sent and received by the Corba ORB and distributed to the other participants. Synchronization of these lists is completely handled by Corba and transparent to the user. Transmitting the object list data with Multicast requires the conversion of the parameter list data for transmission and the unpacking at the receiver's side which is implemented in the class t_Buf and t_Convert. Because we use our own protocol MBP, we have the possibility to couple other applications to the MRT-VR.

Objects are reconstructed using a parsing mechanism. By calling the parser with the stored (changed) object parameter list the object is returned and inserted in the scene, while the old object reference is zeroed. Objects should not be deleted, because they may be referenced by other container objects leading to the need of an efficient garbage collection or reference counting mechanism.

Objects are reconstructed in the following order: lights, shader and objects. While objects are update as long as CHANGE_UPDATE flags are set.

A problem arising in the object list that has to be treated especially is multiple object references. It is useful to use object instances several times to reduce memory consumption leading to multiple object references. When replacing an object with the conventional delete/insertion cycle the object address is changed and has to be provided to all containers referencing the object. This can only be handled by providing a back reference list for this object which leads to a change of the scene containing the referenced object. If a reference cannot be resolved, e.g. when a shader is deleted and no new shader was assigned, the object referencing this object is not deleted, but the missing element is exchanged by a dummy element (e.g. dummy shader).

Though the interface provides object modification for simple MRT objects, it is recommended that multiple object changes are not implemented using the transformElement() method on simple MRT objects, because of the great amount of time and space wasted by this way of object modification. We suggest to use reference objects pointing on the object to be modified instead.

A special handling is provided for reference objects. Reference Objects implement dynamic behavior to the static MRT scene graph Structure allowing an easy transformation of a given object. This mechanism enables object modi-

fication without the disadvantages of an object deletion/insertion cycle. So, reference objects are treated specially in the object list. If a reference object is modified, e.g. transformed, no reconstruction process is applied. The transformation is passed to the reference object only, allowing a fast modification of objects, because the transformation is applied to the object at rendering time, as known from common scene graph API's as OpenGL or Performer.

3.3 Data Transport

Data transport was realized as a collection of different data-transport classes. These classes realize connections between each other using different protocols like TCX, UDP, Corba or M-Bone. The modification of the scene-database is realized by a set of messages which are sent to the different MRT-VR's using the data-transport classes. These messages encode the operations which should be executed by the different programs. To realize distributed environments, messages have to be passed to every participant resulting in an exponential increase of messages. Even a small number of participants may cause problems when using one ore more servers [Pul96][MB94]. To solve the problem of transmitting a huge amount of data to a huge amount of participants and of assuring that important packages are really transferred is a difficult goal and still a matter of research. By separating the messages in important and unimportant messages we can see, that there are only a few important but many unimportant messages. Important messages are concerning geometric object states such as locking, modification, inserting or deleting objects. Changes of viewpoints are less important but very frequent.

Examples:

- a distributed presentation consists of a data transmission phase at the beginning and mostly viewpoint changes or picking information during the session.
- a distributed simulation (e.g. robotic experiment) consists of a data transmission phase at the beginning and many viewpoint changes. Packet loss of the robot position information is justifiable.
- a distributed multi-user edit environment needs a data transmission phase at the beginning or when a user joins the session and several object modification events while editing. Information loss is not acceptable. Updates concern one object at a time, resulting in low data rates using secure transmission.

During the research phases we implemented different transport mechanisms using different protocols and server architectures.

3.3.1 TCP/UDP and TCP/IP

As a testing environment a common socket-based-transport mechanism has been implemented. It supports only 1:1 connections. This module can be used if a one-on-one communication is sufficient. The TCP-module can be used on SUN-OS, MIPS and windows systems. As an extension a TCP router was implemented to support 1:n connections. This system was operated only for testing purposes and showed poor quality for high data rates. It can be useful to transmit messages to all participants at low transmission rates, e.g. for CSCW of small user groups. The TCP-class supports byte transport only.

3.3.2 TCX

TCX [Fed94] is a TCP-based communication protocol developed for robot control. TCX usu-

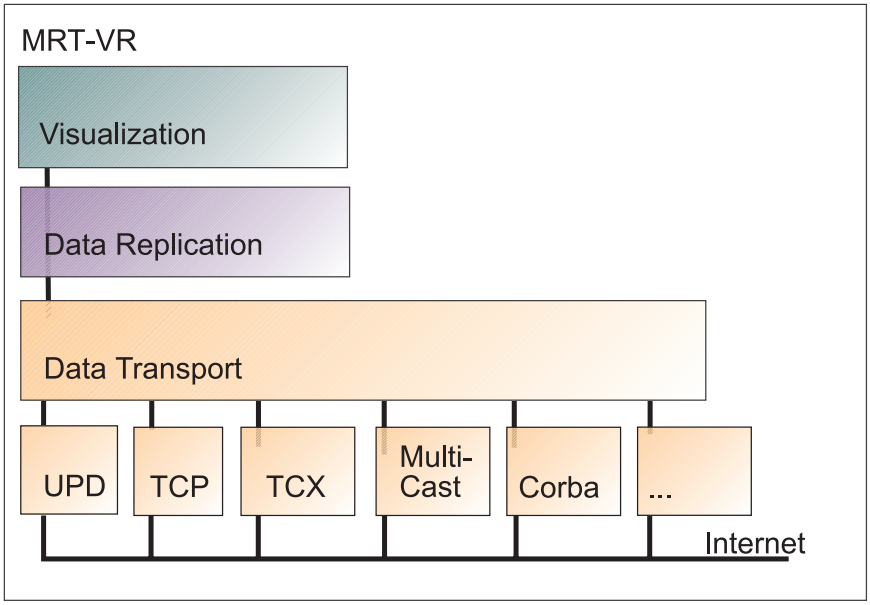

Figure 3.5: MRT-VR Data-Transport-Layer with Different Transport Modules

ally uses one server which handles data connections. Different extensions were implemented to support 1:n connections using a subscription metaphor. TCX supports message typing which allows the use of structs instead of bits and bytes. TXC is sufficient for small user groups known from CSCW environments. TCX servers are easy to setup and to maintain and are available for multiple platforms such as SUN-OS and MIPS. TCX modules were used to establish connections between robotic simulations and the MRT-VR.

The message typing mechanism in TCX led to our abstract message class, which allows to send and receive data by the use of a buffer class t_Buf. Data can be passed to or from the buffer using different types of data like integers, bytes, floating point numbers or character strings. IEEE type conversion and byte reordering is implemented in the buffer class simplifying network data access.

3.3.3 M-Bone

With the use of M-Bone [MB95] connections, data automatically reaches all participants by distributing it to the network once. This reduces the amount of data processed from $O(n^2)$ to O(n). M-Bone-applications have been developed for transmission of video, audio and many other types of data. Multicasting does not assure secure connections, and there may be some lost packets. We use the Real-Time-Protocol (RTP) for Multicast communication which provides information about sender and transmission quality. A reliable transport based upon Multicast is not implemented. So lost messages stay lost.

3.3.4 Data Transport by Corba

In [Leo99] we stated that the implementation of a distributed database system requires two buffers. A write buffer to handle scene changes and push them to the network and a read buffer which receives the data from the network. Both buffers are implemented in the class

t_ObjectList. This mechanism is called "multi version locking". A reading of an object state is realized by using the read buffer, which minimizes synchronization overhead but may led to read an old object version. A change of an object state is always performed on the write buffer, which provides the actual state assured by synchronisation with the server.

Another advantage of this mechanism is the local locking management. A copy of all locks is passed to the participants, so that they can decide individually, whether a locking of an object should be allowed. Only if the local copy of the lock-list allows a locking of an object, the server is contacted to set the lock.

A two version buffer of the object list has another advantage: it allows an easy implementation of a transaction buffer. When starting a transaction all elementary object changes are applied to the write buffer. If the transaction is committed, all changes can be forwarded to the read buffer leading to a scene update. If the transaction fails, the write buffer can be reconstructed with the entries in the read buffer.

When calling the updateWorld() method of class t_ConfInterface the write buffer is emptied and transmitted to the server where it is passed to the other participants leading to changes in their write buffer. The update of the read buffer is initiated by the application, when possible at idle time or by a timer message. The object list performs the necessary changes to the objects.

Corba messages reach the application as "Command" messages initiating the change of elements. A message from the application to other participants is called a "Multiplier" message.

The class t_ConfImplementation handles these messages, calls the appropriate methods of class t_ObjectList and performs the changes or sends the messages to the other participants.

3.3.5 Data Transport by Multicast

Equal to Corba the Multicast messages have to be passed to every participant. In contrast to Corba, the RTP protocol does not provide any information, if a message has reached all participants. Therefore, it is not suitable for reliable communication but very suitable to push the actual camera information to the participants. This is even more suitable in a distributed multi-user system, because camera changes are very frequent and therefore would be a heavy load for a server. On the other hand, a loss of a camera message is acceptable, whereas a loss of an object lock is not. Extrapolation of camera positions has been implemented for the robotic experiment. By locally simulating the robot's movement its position (and therefore the robot's camera position) could be extrapolated when transmission was interrupted.

All messages which are sent through Multicast are coded as MRT Binary Protocoll (MBP) messages. Camera changes are passed to the session calling the changeCamera() Method of class t_ConfInterface. The actual camera parameters are read out of the camera object and packed, using an overloaded pack method of class t_Convert, into a camera CHANGE_CAMERA message defined by the MBP. This message is sent using the RTP protocol. The camera thread of the Multicast implementation handles the transmission and receipt of Multicast messages. The thread is timer controlled and polls the Multicast buffer for new messages several times a second. MBP messages are sent in the same interval. The camera thread also handles the sending and receiving of the RTCP messages which are used to calculate statistics and to provide information of the transmission quality. The RTCP messages are also used to send the address of the current Corba server (IIOP) so that new participant can easily access a session.

At a session login the client can listen to RTCP messages and extract the IIOP address of a currently running session to connect to this server.

Figure 3.6: MBP Message Structure

When a camera change has been received, the parameters are unpacked, and the camera is updated in the user list. When the user for which the CHANGE_CAMERA message has been received has loaded an avatar, the avatar is positioned at his camera position. This is done locally by changing the current transformation of the avatar scene. The change of the avatar is not registered by the Corba session, because this would lead to a scene update for all participants and to a great amount of Corba messages.

To prevent occlusion for the current camera, the avatar of the currently selected view is invisible. Non camera messages are unpacked and passed to the object list to generate the equivalent scene changes. Equivalent to the Corba implementation an "update world" command collects the scene changes and broadcasts update messages to the participants.

Coupling a Simulation

With the use of the MRT Binary Protocol MBP one can control objects within the VRML scene or the state of MRT-VR viewer to use MRT-VR as a specialized display server. Results of a simulation can be sent to MRT-VR causing the MRT-VR to show the simulation results. So, there is no need to generate a visualization front-end [Fel96]. A scenario with a robot experiment using this technique was performed between the University of Bonn and the University of Dortmund is described in Section 7.

Allowing a connection needs an identification mechanism for the objects to be moved. MRT-VR provides two versions of connections for external simulations. These two methods are based on different layers of the MRT-VR system.

The first method is based on the transport mechanism, whereas the second mechanism is based on the data replication mechanism.

Using the MBP

Using the transport mechanism the simulation software (called simulator) can make use of the MB Protocol and use the MBP messages to introduce new or modify existing objects in a scene. Modifying existing objects needs a reference mechanism for these objects, which may be described in a VRML or MSD file. To identify objects, we can use the VRML's DEF statement or a newly defined object statement when using a file in MSD syntax.

To couple an external simulation to MRT-VR we need a model of the real environment as a VRML file. The objects which have to be modified within the virtual environment should be defined by a DEF command. Each object or sub-scene, defined by DEF becomes a modifiable sub-scene with its own (remotely accessible) transformation matrix.

```
DEF _OBJECTRHINO
   Separator {
     Translation {
        translation 13.4 .6 -9.41
     }
      Rotation {
        rotation 0 1 0 1.623
     }
     DEF MatRhino Material {
        diffuseColor   .0 .0 .1
     }
     Cylinder {
          radius 0.3
          height 1.6
     }
}
```

Figure 3.7: Example for a VRML File

Using the MSD scene description language the define statement looks like this:

```
object  cactus = include cactus.inc;
EXT CACTUS cactus;
```

Figure 3.8: Code Example MSD File

Here the first statement loads the include file 'cactus.inc' into a predefined scene, and allows to use this scene as object 'cactus' throughout the whole file. The second statement enters the object cactus with the name CACTUS in the list for external object references.

MRT Binary Protocol MBP

The MBP is a binary protocol to exchange data between MRT-VRs or to couple a simulation with MRT-VR. The MBP is used to send commands and data via messages. A typical MBP command consists of a 64-bit tag, a 16-bit length information (header+data), and the data. All values are coded in network order. To assure backward compatibility length information is enclosed to skip a packet with an unknown tag. It is allowed to send a couple of commands in one packet.

Each message class defines its own tag, and data format, which is described in [Hop97].

The modification of an object is coded as:

- *tag: double, 64 bit, value: modify object: 0x02050000000000000*
- *length: (int,2 byte, Value: 78)*
- *objectID : (long, 4 byte, value: 0001)*
- *transformation : (4x3 matrix of floats, 12 times 4 bytes)*

The modification of a camera is coded as:

- tag: (double, 64 bit, value: modify camera: 0x0203000000000000)
- length: (int,2 byte, value: 58)
- eye point (float,12 byte, value: 3D vector)
- look point (float,12 byte, value: 3D vector)
- up vector (float,12 byte, value: degree)
- HorFieldofView (float,4 byte, value: degree)
- VertFieldOfView (float,4 byte, value: degree)
- orientation (float,4 byte, value: radian)

(A user ID to recognize the user comes with the RTP and RTCP data packet)

Using the Data Replication Layer

The second method is based on the data replication. Because every application taking part at a session has access to scene data, it can easily add new scene content or edit existing content. To identify objects in a given scene the above mechanisms can be used. The named objects are stored in a Name-Space in the object t_FullScene. Object or camera positions can be retrieved by name and used to change the object or use a predefined camera position. A change of a single object, camera or light has to be distributed to the other participants calling the appropriate 'register' function (See Chapter 3.2 for details).

If you don't want to retrieve objects from a VRML or MSD file, new elements could be inserted in the scene via the program API, keeping the pointers to reference these objects in a local structure. Modification of the objects, e.g. transformation or translation is done on the local copy. Calling an appropriate 'register' function will propagate the changes to other participants.

To speed up object transformations and minimize communication with a server or other participants, a bunch of messages can be

bundled via a startTransaction() and stopTransaction() method. Starting a transaction which fails to lock an object, rolls back to its previous state. This state is reconstructed from the scene buffer. All locks are fetched at the beginning and released at the end of a transaction to avoid deadlocks.

Figure 3.9 shows a screen-shot of a 3-cylinder motor model, visualized by MRT-VR. The model consists of 15 parts with approximately 30,000 polygons. We simulated the function of this engine via a small application which calculates cylinder movement and crankshaft rotation. The transformations were applied to the references of the motor parts and registered by the simulation application. This led to a propagation of the scene changes visible by other participants of a session. The objects were identified by named references in the VRML file. Using the names the system returns the pointers to the reference objects, to which the transformations were applied.

3.3.6 Selecting the Actual Transport Mechanism

The application programmer can decide which transport mechanism should be used for a special message. A session is accessed through the session interface. During the instantiation of the session the underlying transport mechanism has to be specified. So, a Corba session uses a Corba transport mechanism, while an RTP session uses a Multicast RTP transmission as a data transport layer. Different sessions can be used at the same time, sharing the same object list. So both sessions can be used in simultaneously providing different transport mechanisms. If an object is registered at the Corba_session, a reliable transport is assured, while when calling the RTP_session register method an unreliable transport is chosen.

When checking the sessions for updates by a call of the updateWorld() method, every session has to be checked for scene changes.

3.4 The MRT-VR System

Built upon the MRT-library the MRT-VR Application provided the framework for user session management and user interaction. During our research we implemented MRT-VRs based on the Microsoft Foundation Classes (MFC) running under Windows 9x/NT and MRT-VR applications running under Unix. The MRT-VR application provides file handling to load and save a scene and different interactors to move in the 3D environment. It also provides session administration like joining and leaving a session or selecting the viewpoints.

Navigation Support

User navigation in 3D space is a difficult task. As showed in [FJ96] users very easily lose orientation, either by stepping 'into' an object (e.g. and diving into the ground or ending up between two sides of a wall) thus, making the reset-to-original-viewpoint button the most frequently used one. To give a more realistic feeling of the scenes, we have implemented several levels of collision detection which directly affect the camera path and thus, meet the user's intuitive expectation of solid objects. Further controlling the user's camera path by methods of distance control and collision detection introduces a consistent way to 'walk' on the ground (i.e. keeping the distance to the ground at a constant value), including the climbing of stairs and averting the jump on tables.

3.4.1 Collision Detection

MRT-VR uses the object-oriented data structure of MRT to represent all VRML objects. Conse-

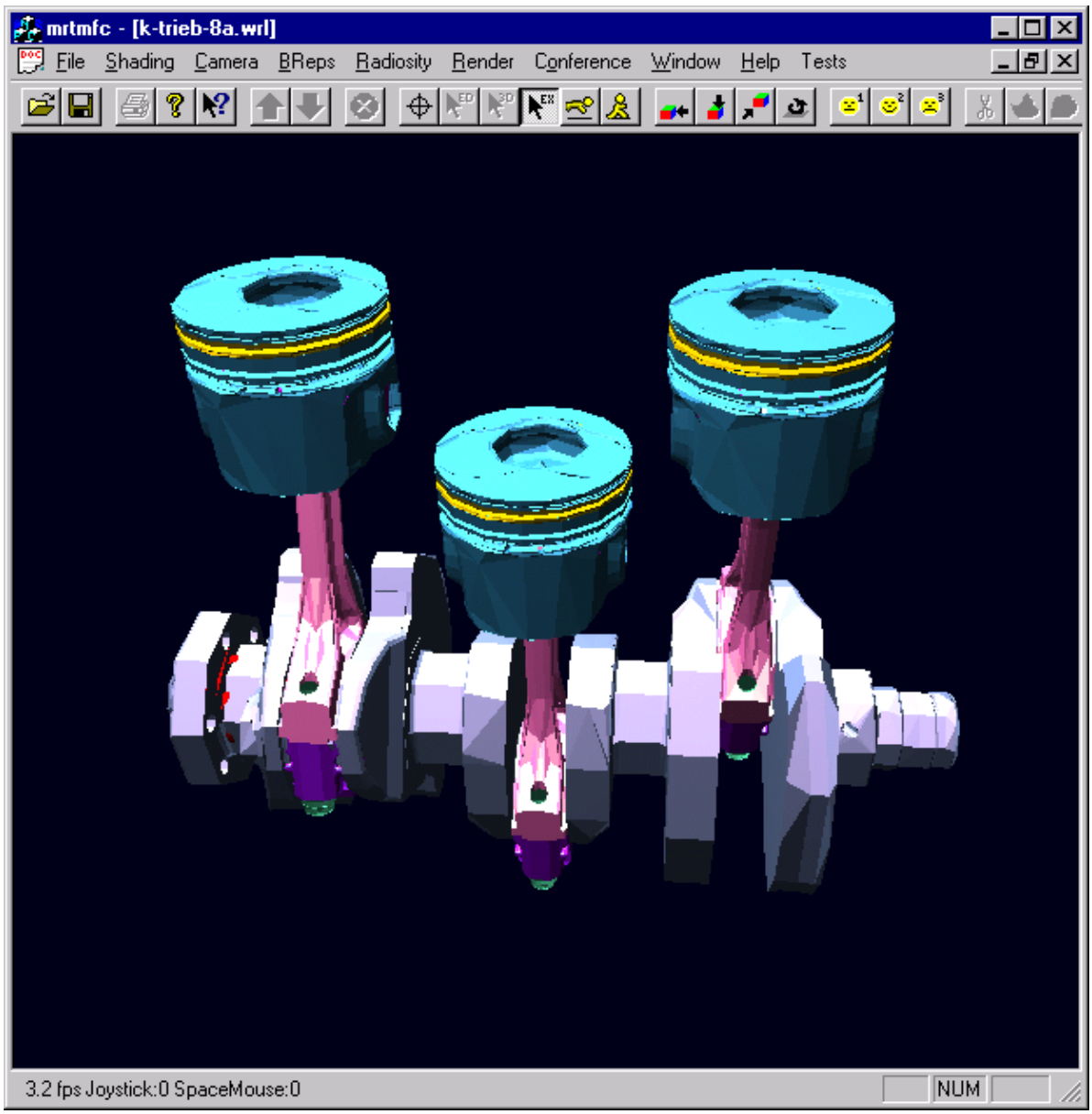

Figure 3.9: Simulation of a 3 Cylinder Motor (Model from Volkswagen AG)

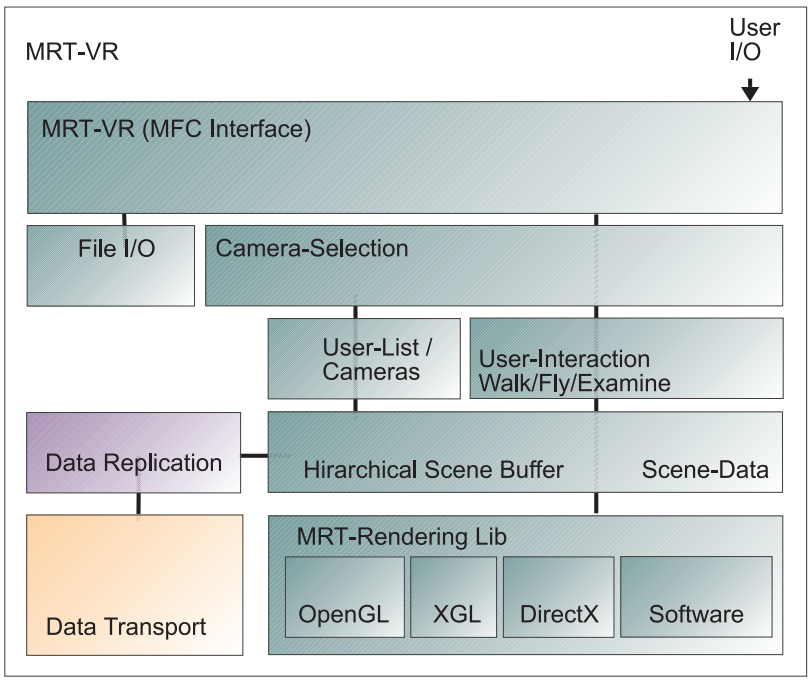

Figure 3.10: MRT-VR Application

quently, collisions can be detected by using the ray-tracing functionality of MRT. The built-in ray-tracing modules compute the intersection of a given ray with a scene, returning the distance to the closest object hit.

So, when the user moves into a given direction MRT-VR casts a ray along the trajectory and checks whether the returned distance is less than a given value. If is the case that the user is about to run 'into' an object and MRT-VR takes the normal vector of this object, the intersection point of the trajectory and the object to determine how to change the user's direction in order to avoid a collision or disappearance.

In having taken care of the collision problem the moving on level ground can now be mastered by most users without significant problems. But navigating on a slope or climbing stairs adds an extra degree of navigation complexity and typically gives the user a hard time. An easy solution of keeping the user at a constant distance above the ground has been implemented in MRT-VR by casting two rays onto the ground: one ray before and one ray after each step. Both intersections return a distance. If the difference is zero, the user is determined to walk on flat ground and no special action is taken. If the difference is within a small range, the user is considered to take a step and the difference is added to the position of the user. If the difference is large and negative, the user is considered to fall and the position of the user is decremented in several steps to the new level, giving the user the impression of a fall. If the distance of the new position towards the ground is too large, the user is either set to fly-mode or just stopped (depending on some preferences). If the user is in fly-mode, the casting of rays towards the ground is used to automatically switch back to walk-mode when the user lands on the ground.

3.4.2 Fly Interactor

A fly interactor is integrated by using two different modes of operation. Flying can be controlled via keyboard or via mouse pointer. When flying by keyboard a key mapping allows flying without collision detection parallel to the camera axes. This mode allows the traversal of a building or the elevation to different levels. When flying by mouse the mouse, screen(x,y) coordinates are used to "fly" towards the indicated direction and the left and right buttons are used to accelerate and de-accelerate. An optional middle button can be used to stop. Intersection testing, used similar to the walk mode, avoids collision with objects and routes the user around the obstacles. A mouse position near the screen center leads to a straight flight in the direction of the look-at point, whereas pointing to the other screen segments a turn in the direction is calculated. The maximum turning angle is half of the aperture of the camera's horizontal field of view (h-fov) and vertical field of view (v-fov) and multiplied with the actual speed value. Thus, for a hard turn the user has to slow down, which is very similar to reality.

3.4.3 Pointer Interactor

When demonstrating objects to a distributed public it is recommended to have a 3D pointing device as part of the scene, because the remote audience does not see a screen cursor. A 2D Cursor, which is at the same position as a mouse over the screen window, may be sufficient when the viewpoints of the participants are the same. A 3D pointing device has further advantages: thus it can point along the normals of a hit object it is able to amplify the 3D impression of an image. When hitting an object with the mouse the pointer points along the surface normal in direction towards the object. If no object can be found, the pointer points to the scene center. Using this feature the 3D cursor can be used to probe surface normals. To avoid the pointer from disappearing into the object when the normals are incorrect, additional tests have to be implemented. So, the cursor may change its shape when a normal is pointing in the wrong direction.

A pointer is allowed, if a connected user is allowed to have an avatar as well. So, in a lecture environment only the teacher has the right to have a pointing device and an avatar. Additional rights may be granted to selected users in run-time.

3.5 Usability Enhancements

To enhance the usability of MRT-VR as a multiuser environment, several changes had to be implemented. Some are used to ensure interactive frame rates, others to allow smooth camera transitions. Other enhancements were made to give an overview of the scene hierarchy or to improve user interactions.

3.5.1 Integration into SDR

As a part of the VR-LAB, MRT-VR should be usable as easy as possible. Therefore, the program was integrated in the common Session Directory Toolkit (SDR) as illustrated in Figure 3.11. SDR allows to participate in Multicast sessions, which are advertised via the SDR Toolkit [Han96]. Users can join an advertised session by a mouse-click and the SDR tool starts the appropriate programs. Up to now most transmissions have been using a video and audio link (VIC,VAT) and some have been using the Whiteboard. From now on it is possible to use MRT-VR as well.

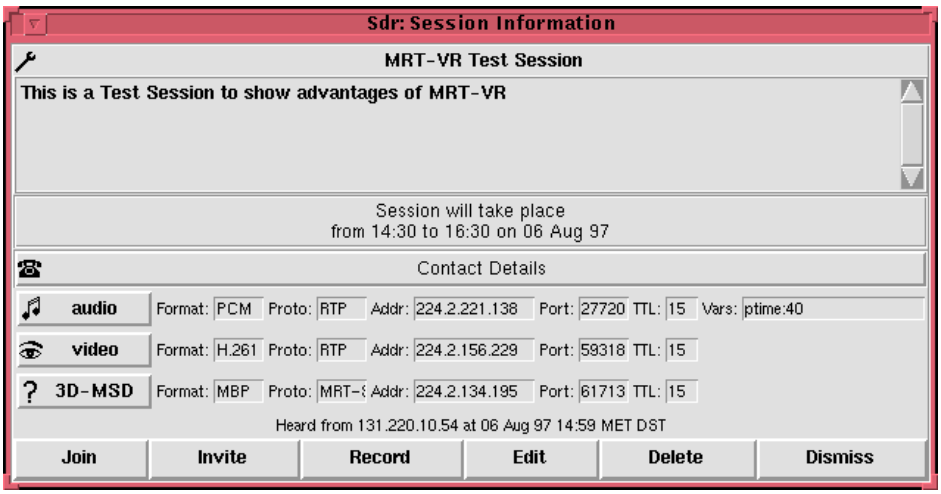

Figure 3.11: Integration of MRT-VR in SDR (Session Directory)

3.5.2 Camera Interpolation

When moving in 3D-space sometimes a jump, a downfall or a camera transition is necessary. To generate a smooth transition between different viewpoints, a camera interpolator was implemented. Based on two different viewpoints a number of intermediate positions is linearly interpolated. The number of steps depends on the distance between the two points and the last calculated frame rate. Frame rate detection is used to keep frame rates interactive, while interpolation calculations are used to keep movement as smooth as possible.

3.5.3 Coordinate System Determination

Determination of the main axes of a given scene is necessary to generate camera transition paths to look at the scene from the front, the top and from the sides. Because in a VRML scene the coordinate axes are not fixed, any coordinate system (x,y,z) or (x,z,y) is possible. Determination of the up-vector (parallel to a main axis) is generated via a heuristic from the camera parameters and the scene dimensions. The heuristic assumes that the user has placed a camera in front of the scene, with an up-vector of the camera pointing in the direction of the scene's up-vector. By comparing the angles of the camera's up vector with the coordinate axes the corrosponding up-axis is found. The look vector, calculated from the difference between look-at point and eye point, is compared to the other coordinate axes to determine the front and right axes of the coordinate system. Of course this heuristic does only work correctly when camera angles are less than 45 degrees to the main axes.

3.5.4 Frame Rate Detection on Startup

Depending on the user machine MRT-VR can select a rendering algorithm and the quality of the VRML model depending on the speed of the underlying graphics hardware. This is based on a heuristic method which assumes that some rendering methods are more time consuming than others and rendering of less detail is faster than rendering more details. MRT allows to select an approximation quality of the model by using each objects approxShape() method. The

Figure 3.12: Avatar Selection

parameters provided with this function allow a detailed quality control of the model. Besides this, the system provides a relative quality parameter, and therefore subsequently increases or decreases the approximation quality which is of course directly related to rendering speed.

At startup the system renders a few images of the scene in different qualities and measures rendering time. The aim of the system is to adjust to a frame rate of 10fps.

The heuristic starts rendering with a standard quality and is switched from flat shading to Gouraud-Shading. This process is stopped when rendering takes more than one second. If one of the methods provides a more or less appropriate frame time, rendering quality is adjusted to increase or decrease image quality while keeping the frame rate greater than 5 fps.

3.5.5 Avatar Selection

In a virtual environment an avatar represents a user's position. MRT-VR allows the user to load an MSD or VRML file as an avatar representation. When meeting in virtual space it may be necessary to change the avatar to express a special state or intention and make this visible to other participants. Therefore, MRT-VR allows the user to load a set of avatars in the scene, out of which he can select one by the press of a button. We defined three buttons to express user states like 'sad', 'happy' and 'normal' (See Figure 3.12. These default switches can be used to recall any avatar described by an MSD or VRML file.

MRT-VR provides an API method onLoadAvatar(), which opens a file selection dialog and asks the user which type of avatar to load. Currently, this dialog supports the avatar types described above but may be enhanced in the future.

Chapter 4

Enhancements of the MRT Library

In the following section we describe the enhancements of the MRT-Library which were added to implement MRT-VR. After this we give a short introduction into the MRT-Library, which is a summary of [FS96].

4.1 Enhancements of MRT

The MRT-Library proofed to be very useful as a tool for rapid developement of 3D applications, based on ray-tracing or approximative rendering. For a distributed application like MRT-VR the flexible structure of the object oriented MRT-Liba made could be easily enhanced.

4.1.1 Parsing Mechanism

Some of the enhancements affected the parsing mechanism, which was necessary to construct new objects at the client's side. The parsing mechanism implemented in MRT was used to ease the construction of new objects and eliminate the need to change a parser grammar when a new object was added to the library. A parser mechanism [Fis96] was generated to allow registration of object constructors and their parameters at runtime. A basic idea of this mechanism is the use of a parameter list to pack typed parameter entries into it to develop a uniform interface. Figure 4.1 shows the parsing system in more detail. The parsing system consists of a parse function which creates the new object, an object definition which returns information on parameter types and their descriptive text and the number of minimal and maximal used parameters if more than one constructor is available. The method objectVar() of each subclass of class t_Object registered the object definition at the parser data at run-time.

In addition to the parser system for automatic object creation we need uniform functions to get the current object parameters. The method writeObject() of each subclass of class t_Object returns all constructor parameters via a parameter list. An optional reconstruct() method has to re-calculate construction parameters from stored object variables of class t_SurfaceObject, because most surface objects do not keep the construction parameters after they were initialized. By using writeObject() while traversing the object tree of class t_FullScene all parameters of all objects can be retrieved and, for example, written to a file. The method writeObject() was implemented for class t_SurfaceObject, t_Shader, t_Lights and t_Camera.

Figure 4.2 shows the code listing for the construction elements for the class t_Sphere

4.1.2 Control Objects

Introducing control objects which embed their control functions in the constructor, the system of remote object construction can be enhanced

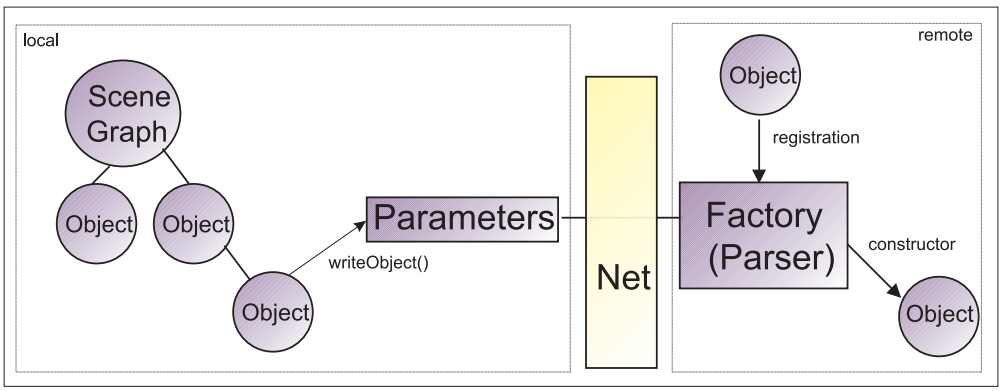

Figure 4.1: Parsing System

to a system of remote function execution. Implementing the functions to delete a specific object in the MRT scene graph creates a 'delete object'. In the same manner the highlighting of objects can be realized via a 'highlight object'. It is obvious that these control objects need information about their environment (e.g. t_Scene) to operate. These parameters occur in the parameter list, but are not sent via the communication link. Parameters of these special types are transmitted as NULL and replaced by a local (e.g. scene) pointer. This replacement mechanism is implemented in the 'parse' function of the dispatcher. The dispatcher is part of the Corba and Multicast transport functions and handles the incoming messages.

4.1.3 DistribID

Because runtime type ID generation produces different results on different computers, constant object IDs have to be used to identify the objects. We extended the class t_ObjectDefinitionMRT for this distribID. In the example above C_OBJECT_SPHERE is the constant for the ID type. Distrib IDs are predefined constants included via the *comdef.h* file. A DistribID consists of a 4 byte integer value. The first byte indicates the return value. A "zero" indicates no return value, which is used for control objects, and a "one" indicates objects of class t_MRTBase, other values are for future use. The second byte indicates the command type (create, modify, delete). The last two bytes indicate the object type, like t_Sphere. When extending the MRT library by a new object, this object has to provide the methods writeObject(), objectVar() and objectParseFunc(). To identify the new object on other systems, a new distrib ID has to be generated and provided with the object in the objectVar() method. Currently, these IDs are generated by hand. A possible mechanism to generate these IDs automatically is to use the ASCII name of the objects and the Corba server to register and distribute objects by their ASCII name and return a new ID for each new registered object.

4.1.4 Reference Objects

To provide object movement, extensions to the MRT library were necessary. The MRT library did not provide efficient methods to move objects. These restrictions resulted from the design goals of MRT. To speed up ray-tracing, the objects are constructed in world coordinates. Object movement can only be implemented via ob-

```
t_MRTBase* t_Sphere::objectParseFunc (
   const t_ObjectDefinition::pStackIterator& param,
   unsigned, const t_4x3Matrix& trf)
{
   // simple approximation of the Euclidean associated norm of a 3x3-matrix,
   // exact calculation would require the eigenvalues of the matrix
   static t_Real trfNorm2 = (t_3DVector(1,1,1) * trf.transformation()).abs()
                            /sqrt(3);
   t_SurfaceShader*   shd = dynamic_cast<t_SurfaceShader*>(param[0]);
   t_3DVector         cen = *dynamic_cast<t_3DVectorMRT*>(param[1]) * trf;
   t_Real             rad = t_Real(*dynamic_cast<t_RealMRT*>(param[2])) * trfNorm2;
   return new t_Sphere(cen,rad,shd);
}
const t_ObjectDefinitionMRT& t_Sphere::objectDefinition ()
{
   static const type_info* objectId[] = {
      &typeid(t_SurfaceShader),
      &typeid(t_3DVectorMRT), &typeid(t_RealMRT)
   };
   static const char* description[] = {
      "surfaceshader","center(lc)","radius(lc)"
   };
   static t_ObjectDefinitionMRT def("sphere", 3, 3,
                                    objectId, description, objectParseFunc,
                                    C_OBJECT_SPHERE, &typeid(t_Sphere));
   return def;
}
const bool   t_Sphere::objectVar =
   t_ParserDataMRT::instance().registerParseFunction(t_Sphere::objectDefinition());

void t_Sphere::writeObject (t_ObjectDefinition::pStackContainer& param,
                            unsigned& numParam, t_4x3Matrix& mat)
{
   mat = t_4x3Matrix::identity();

   // assign return parameter
   numParam = 3;
   param.push_back(shader());
   param.push_back(new t_3DVectorMRT(center()));
   param.push_back(new t_RealMRT(radius()));
}
```

Figure 4.2: Construction Elements for the Class t_Sphere

ject deletion and reconstruction at a different position, which consumes much memory and time. These limitations lead to further restrictions for large models, because the use of world coordinates avoids the re-use of object data, especially when using an approximative renderer. When approximative rendering is necessary, like in VR-applications, all objects have to be represented with a quad or triangle mesh, which is generated by the scene graph objects when calling the object's approxShape() method. Avoiding referencing laeds to an extensive amount of memory used to generate objects meshes.

On the other hand, the introduction of a complete local coordinate renderer, using uniform coordinates for each object, is too expensive for ray-tracing and intersection tests. Also, hardware accelerated graphics are very fast in re-rendering pre-calculated display lists and in avoiding the excessive use of matrix multiplications in the rendering process. Keeping in mind that in VR-environment rays were used to calculate intersection tests for collision detection, we will easily see that there is a trade-off between fast ray intersections and fast approximative rendering.

In a virtual environment we often find a lot of fixed objects and only a few animated objects. This encourages the introduction of reference objects.

A reference object is an encapsulation of a normal t_SurfaceObject. A reference object holds a pointer to its referenced object (e.g. a sphere), a pointer to a shader and a transformation matrix. Additionally, bounding volume calculations and intersection tests are

re-calculated taking the current transformation into account. Using a geometric object constructed in uniform-coordinate-space a reference object can be used to transform it to any scale, rotation and position in the world coordinate system. Further extensions of the MRT library allow the exchange of the referenced object's shader by the reference object's shader and the re-calculation of all intersection tests sent to the reference object. Using a reference object pointing to a scene allows the easy movement of complex scene elements.

One single static scene can be rendered by using a single display list, allowing very fast approximative rendering. Introducing reference objects the rendering engine has to be extended when using display lists. The display list management has to manage several lists for different objects. One strategy is to use one display list for every t_Object normally consisting of a great amount of quads or triangles. Differencing between object geometry and object shader was necessary to provide refobjects with exchangeable shaders. Of course, this overhead leads to a slower display of the approximative rendering. But on the other hand, it enables scene and object based hierarchical view frustum culling [MF99]. Allowing to view scenes of a complexity not been able to be rendered with the MRT library before.

The class t_Refobject introduces different methods to translate and transform a given reference object. So, lc2wc() returns or sets the transformation matrix for a given object, while wc2lc() sets or retrieves its inverse. The inverse is calculated on demand. A fast multiplication algorithm was used to minimize overhead when multiplying matrices. The translate() method moves an object relative to its position, while transform() multiplies a 4x3 matrix to the actual *lc2wc* (local coordinate to world coordinate) matrix.

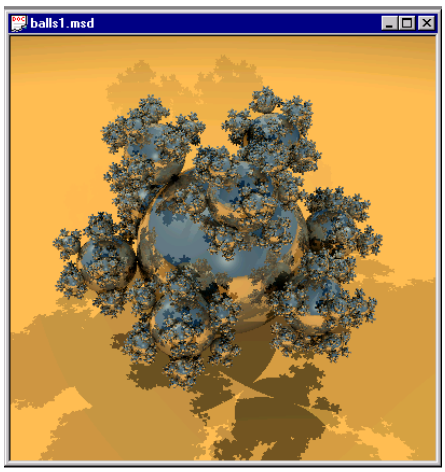

Figure 4.3: Sphereflake, raytraced image, 7384 spheres

Introducing reference objects was necessary to render very big scenes, as shown in the pictures 4.3. The first picture shows the raytraced image of this scene containing 7384 spheres. When generating the approximative image, 590562 polygons have to be constructed for an image in medium resolution, leading to the creation of the same amount of vertices and normals. By the use of refobjects only 7384 refobjects and one sphere with 80 polygons have to be created.

As stated before, the ray-tracing of a scene containing references is expensive, due to the amount of matrix multiplications necessary when intersecting rays with the scene. To avoid this, the class t_Scene was provided with a freeze() function, eliminating the reference objects, leading to the construction of new objects, without a boundary representation. Each reference object is supplied with a visibility() flag. The visibility flag allows to control, if a referenced object or scene should be rendered or not. Using many reference objects in one scene allows an easy implementation of a 'level of detail' (LOD) function. A set of reference objects is used to select a user representation (avatar) in MRT-VR. Reference objects can be

used hierarchically.

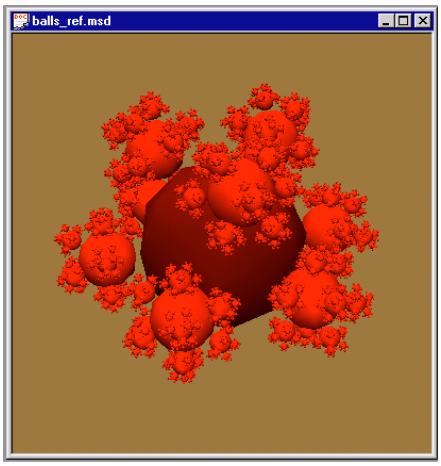

Figure 4.4: Sphereflake, Gouraud shading, 590562 polygons

4.1.5 Parser Extensions for Coupling of External Simulations

MRT-VR allows external simulations to connect to the renderer. So MRT-VR can be used as a display engine for scientific calculations. Allowing a connection needs an identification mechanism for the objects to be modified. MRT-VR provides two versions of connections for external simulations. These two methods are based on different layers of the MRT-VR system.

The first method is based on the transport mechanism, whereas the second mechanism is based on the data replication mechanism.

Using the transport mechanism communication is done via the MBP (MRT Binary Protocol) described in Section 3.3.5. Objects are identified by name, so the objects to be moved have to be labeled. In a VRML file we can use the DEF statement, while in the MSD scene description language we introduced named objects.

To support naming of objects and cameras in VRML and MSD files, the corresponding scene grammars VRML.g and MSD.g were expanded. Both scene grammars are created using the ANTLR parser mechanism described in [Par95]. The class t_FullScene was expanded to reference the list of cameras defined in the VRML or MSD file.

In the VRML grammar we use the DEF statement to name objects. Because VRML allows to use any node type for the use in DEF and USE statements and supports no type information, we have to identify the objects type by its name. Though all object parameters can be modified having an objectID it is sufficient to identify objects and cameras.

The "DEF _OBJECT'" statement identifies a geometric object. When defining a external reference with the statement mentioned above the object can be referenced by its name. See Figure 3.3.5 for sample code. Figure 4.1.5 defines a camera with the name MAINHALL to be used as an external reference.

The MSD grammar was expanded to support object definitions. Objects are a type of variables like integer or real numbers. An object can be defined like a variable and used to instantiate a new reference. By default, object definitions are NOT painted, because they are normally positioned at the coordinate origin and transformed by a separate transformation matrix. Figure 4.1.5 shows an example for the definition of a re-usable scene object.

Figure 4.1.5 allows to define an include file and to reference its definition by name and its referencing call.

In this case a reference to the sub-scene is created. In case of scenes the color assigned to the sub-scene is ignored. While referencing single objects the color of the definition can be overridden by assigning a new color.

To add an object to the external references list, it should be marked with the tag EXT. The definition of the object cactus, which could be referenced via MBP as the object

```
DEF _CAMERA_MAINHALL PerspectiveCamera {
   position -361.4 165.4 -2799
   orientation 0.9932 -0.1049 -0.04969 -0.8899
}
```

Figure 4.5: Code Example VRML File for Camera Position Definition

```
object subscene =
BEGIN
  SURFACE   4  1.0;  (0,  200, 255) 1.0;    (0,0,0)    .0,  1;  ( 0, 0, 0)  1
  SURFACE   5  1.0;  (0,  220, 225) 1.0;    (0,0,0)    .0,  1;  ( 0, 0, 0)  1
  SURFACE   6  1.0;  (0,  240, 225) 1.0;    (0,0,0)    .0,  1;  ( 0, 0, 0)  1
  SPHERE 4 (2,0,0)  0.8
  SPHERE 5 (4,0,0)  0.8
  SPHERE 6 (6,0,0)  0.8
END
```

Figure 4.6: Example for an Object Definition, MSD Syntax

OBJECTCACTUS, is shown in Figure 4.1.5. Object definition allows the re-use of objects (instantiation and referencing) via reference objects. When referencing an object the actual transformation matrix is used to translate the object to its destinated position. Figure 9.2 shows the modified MSD grammar.

The VRML scene grammar was extended to support instantiation and references. A global switch changes the default from instantiation to reference. The switches can be set via the t_MRTControl data structure using the commands useReferences(bool) and showDefinition(bool). The first function changes the default behavior of the VRML parser from generation of new instances to generation of a reference. The second function is used to change the behavior of the VRML parser so that the first instance (definition) is made invisible, which is the expected behavior for additional scene geometry used by external simulations (but which is not compatible to the VRML standard). Further the grammar was expanded to support camera's lists for the camera external reference list.

4.2 Architecture of MRT

The *Modular Rendering Tool (MRT)* was designed to serve as a testbed for the implementation of new algorithms and as a tool for teaching [Fel92].

Currently, MRT operates in one of two modes: it either creates photo-realistic images by ray-tracing the scene or polygonal approximations of the scene. These polygons can then be used as starting patches for a radiosity computation or directly be rendered by CGI3D [FFW93], which is an abstract 3D device interface based on the concept of CGI [ISO89]. CGI3D utilizes existing graphics hardware through calls to OpenGL or XGL library functions on Unix systems and through calls to OpenGL or Direct3D on MS-Windows systems, provided, these packages are supported on the actual system.

4.2.1 Objects

Class t_Object is the base class for all types of objects that can appear in a 3D scene. There are several categories of objects derived from this class:

Geometric objects (t_SurfaceObject**)** are primitive rendering object that have a 2D surface, like spheres or bezier patches.

Container objects (t_Scene**)** structure sets of primitive objects spatially or logically into sub-scenes.

```
object  cactus= include cactus.inc;
ref subscene;
ref cactus;
```

Figure 4.7: Example for an Object Definition for Include and Reference Calls, MSD Syntax

```
ref singleobject 5;
ext OBJECTCACTUS cactus;
```

Figure 4.8: Referencing an Object. Referencing and Assigning a New Color, Entering in External Reference List

These classes are described in the next sections in more detail. Their common base class t_Object provides the following virtual methods:

t_BVol boundingVolume() returns the bounding volume

bool intersect() computes the intersection between the object and a ray.

bool checkIntersect() just performs a Yes/No intersection test, which can be done significantly faster than computing another version called t_3DVector surfaceNormal() which computes the normal to the surface at a given point.

bool approxShape() takes a quality control parameter and creates a faceted approximation of the object.

bool mapInvers() maps a point on the surface of that object onto a two-dimensional texture space defined over $[0,1] \times [0,1]$.

t_Shader* shader() gives access to the object's shader.

Geometric Objects

The base class for geometric objects (i.e. solids) is class t_SurfaceObject, which is derived from the classes t_Object and t_Approximation. Class t_Approximation is responsible for maintaining the object's boundary representation. MRT currently supports some 20 geometric objects. Additionally, CSG expressions with arbitrary objects can be built. The resulting CSG objects support ray tracing as well as a polygonal representation, i.e. a BRep.

Object Containers

It is worth mentioning that the whole scene as well as all sub-scenes are handled consistently as elementary scene objects. That means, class t_Scene is also derived from class t_Object and thus also provides the above messages.

Different schemes to accelerate the computation of intersections between a ray and a scene may be realized by deriving new classes from class t_Scene and overriding its intersect() messages.

4.2.2 Rays

Most rendering algorithms require intersection tests between a ray and one or more objects. Not only ray-tracing but also form factor calculations for radiosity, collision detection, shadow calculation or object picking can make use of a ray-casting module.

4.2.3 Shader

Class t_Shader is the abstract base class for all illumination models and gives access to the surface parameters to implement, for example, the Phong illumination model as well as the functions diffuseCoefficient(), specularTransCoefficient(), incidentLight(), and attenuateLight().

Similar to class t_Object, two classes are derived from t_Shader to seamlessly model the interaction of solids as well as volumes with light. These are described in the next sections.

4.2.4 Light Sources

Class t_Light implements a positional light source and serves as the base class for all light sources. The key of defining new types of light sources like spot lights or directional light sources is the virtual function directLight().

Area light sources of arbitrary geometry are supported by class t_LightObject. A light object contains the BRep of a geometric object (Section 4.2.1) and provides the necessary sampling methods. Class t_AttenuatedLight accounts for the attenuation due to transparent scene objects. All types of light sources (point light, spot light, directional light) also come in a distributed version, i.e. they simulate a penumbra by casting jittered sample rays from the light source.

4.2.5 Camera Model

Class t_Camera implements the modified pinhole camera model. After being initialized with the viewing parameters message getRay() returns the initial ray from the observer through pixel (x,y). Different camera models can be implemented by deriving new screen classes, which redefine the messages getRay() and project().

The supported projection models are *perspective* and *parallel* selected at the time of construction or with method projectionMode(). The corresponding projection matrix can be calculated with method calculateProjectionMatrix(). It can also be retrieved from the class for direct use by the application program.

4.2.6 Rendering

Class t_Image is the entry point to all provided rendering algorithms. Method rayTrace() contains the main loop over all pixels for ray-tracing based algorithms. For each pixel to be computed one or more sample rays are launched into the scene. Protected messages like gammaCorrect() and formatOutput() take care of gamma correction and formatting of output, respectively. Adaptive or stochastic anti-aliasing can be performed by derived classes (e.g. class t_AAAImage for Adaptive Anti-Aliasing) redefining virtual message rayTrace().

Message approximate() takes the faceted approximation of the scene created by message approxShape() sent to all objects and displays it with the polygon renderer CGI3D. Currently, supported shading options are wireframe, flat, Gouraud, Phong (with one of the illumination models ambient, diffuse, Phong, monochrome, or depth-cuing). Additionally, CGI3D supports different rendering libraries like OpenGL, XGL or Direct3D, to exploit hardware based rendering on the current platform if available.

4.2.7 Approximative Rendering

To apply a standard rendering algorithm (wireframe, flat, Gouraud, Phong) message approxShape() is sent to the scene causing all objects to create their faceted approximation. Depending on whether the information on the scene's hierarchy shall be used for the rendering process or not (hierarchy versus flat list of objects) MRT offers a different set of utilities to traverse the scene.

Images can be created either statically (just by display of the scene) or together with an interactor allowing the interactive manipulation of the camera parameters. Animations (fly-throughs) can be created by entering an extra file specifying the camera positions and orientations as well as the interpolation mechanism.

Chapter 5

Distributed Presentation Environments

Besides the use of a browser software to access 3D content distributed work and presentation needs a special environment to present distributed media and perform distributed sessions. These presentation environments may be used to record, transmit and playback lectures from or to other presentation environments. Presentation environments extend normal lecture rooms by the availability of different hardware and software. Unfortunately multi-media lecture rooms are at the moment very rare in Germany. A short descriptions of some lecture rooms will give an impression on their capacity.

5.1 Lecture Rooms

In this section we discuss some established media lecture rooms (MMHS). The objectives of this summary are

- supported media
- media quality
- remote access capabilities
- user / participant interaction

5.1.1 IGD Darmstadt

The IGD (Institut für Grafische Datenverarbeitung) in Darmstadt was equipped in October 1997 when the new IGD building was opened. Their MMHS was endowed with two video light channel projectors, three cameras, an electronic overhead projection system, an electronic blackboard, video tape recorders and audio equipment. The front desks and the speaker's desk were equipped with connectors for notebooks or workstations, audio input and power connectors. The operator, situated in the back of the room, was equipped with a media control system, a video mixing unit and some control monitors. The VCR was situated in the speakers desk, the other electronic equipment was stored in an extra room. All furniture was made to order. The constructor wish was to hide an open notebook, led to a very high speaker's desk, covering small speakers.

Video projection was done by two NEC light channel projectors, with a high light output. The projectors were mounted behind a wall, while a double mirror system was used to project the image backwards onto two screens. The screens were approximately 2.5 m wide.

Using back projection has several advantages: first, the projectors are not seen, second, noise is reduced and third, a nice wall provides a pleasant scenario. The disadvantages are the loss of light power, because the light has to pass through the screen and the existence of two mirrors in this setup. A restricted view angle can result from the use of some refractive material for the screen (Fresnel lenses). In the described setup two Fresnel lens screens were used where

the small viewing angle led to a picture of low brightness for viewers sitting in the first rows. The projectors did not seem to be properly adjusted (color, convergence), resulting in a poor picture quality. The mounting of power-supply wires beneath video wires caused interferences with the video images.

Notebook images were converted to video signals and fed into the projectors, by converting the RGB signal to a video output and projecting the video signal resulted in a poor image quality. For overhead projection a Busch video system with an integrated camera was used to achieve a high integration of all components. Camera and lamp were mounted on the same side of the slide producing reflections, additionally the camera could not resolve small fonts on the slides. A great advantage of such an overhead system is the ability to project three dimensional objects.

Video projection was used for 35 mm film slides also. This was done by projecting the slides to a wall in the media room, capturing the image by a video camera, and projecting the image with the video projectors. This resulted in poor picture with bad colors and low resolution.

To capture images of the audience, two cameras were installed in the lecture room. These room cameras were mounted on a remotely operable pan tilt head. Without noise reduction it resulted in a loud hum whenever the cameras were operated, attracting the audience's attention. The cameras (standard industry 1/3"CCD, 10 times zoom) produced dark pictures of the audience, even if the lights were on. Furthermore the zoom range was very poor and did not allow to show a persons head in full screen size. The audio equipment acceptable, though sometimes a cracking sound appeared. Two microphones mounted on the desk provided a good audio quality even if the speaker turned his head away from them.

5.1.2 HNI, University of Paderborn

Recently, in May, 1998, the Heinz-Nixdorf-Institute in Paderborn opened the "first European interactive lecture room". This room was developed to establish a teaching network, which means that every PC screen can show the content of another PC. The n:n mapping is selected by the administrator sitting in front of the audience. These features were used to present working results of one person to all other persons. The presented "shared workspace" was realized using Hyperwave [Mau96] as a CSCW environment to store the work of the students. Internet access was not a design goal of this system, but using Hyperwave, it is easy for external persons to share the working environment. Besides this n:n switch a video tape recorder and a video projector (2x3m, ceiling mounted, CRT system, NEC) as well as remote operable lights, audio amplifiers and some microphones were installed. The equipment can be controlled with a media control system, which allows to control the VCR, the position of the projector's picture and the video source. The system provides five pre-sets for lights which can be selected by the press of a button. A compact disc player can be started and the overall volume can be adjusted. A Simone echo compensator was used to kill local echo.

A Bush overhead system was used in exchange for a normal overhead projector, having the advantages and disadvantages described before. In an interview a technician stated, that a permanent Internet access of the system for a video transmission was not part of the project and could be installed temporarily. No video cameras were installed, because the system was designed to be used in an intranet only.

5.1.3 University of Dortmund, Multimedia Lecture Room

The Distributed Conferencing Project (DISCO) in Dortmund was equipped in 1996/1997 with a Barco projector, an auto convergence adjustment system, a motorized screen, a single remote operable zoom camera and a second fixed mounted camera capturing the blackboard or overhead projection screen. Audio capturing is done using SUN audio microphones. A SUN SPARC computer is used for video digitizing. For data transmission an ATM connection is used to provide a high bandwidth for the resulting Internet video streams.

The system was mounted in a steel case, containing the audio amplifier, two VCRs and the SUN computer. A video crossbar for video selection and a second SUN computer for remote control were mounted in there, too.

The system is able to record video and audio streams to a VCR and playback videos on the projector. High quality RGB output can be projected with the projector. RGB and video systems were not coupled and thus prevent RGB signals to be transmitted via M-Bone tools. The camera is extreme silent not disturbing the people in the lecture room.

Java programs allow to connect to the MMHS to get video or audio information and, depending on remote user's technical capabilities, to talk, show or type back. The complete system (excluding audio) is remote controllable via a Java application, allowing an experienced operator to control a lecture from a remote place.

5.1.4 Discussion

Though the lecture room in Darmstadt was very expensive and a lot of equipment was mounted in there, the resulting quality could not convince. The idea of using the lowest common quality for all media seems to result in an overall bad image quality. This worsened by the poor design of the back projection and the adjustment of the projectors.

In the HNI in Paderborn the monitor n:n selector and the media control systems are not integrated so that there are two control boards appearing on the desktop. The media control system has a hierarchical menu structure, which may lead to difficulties finding the correct menu entry. Because video cameras and Internet access were installed temporarily only, this installation is not capable of performing a distributed session.

In Dortmund the picture of the projector was not satisfying, although an auto-convergence unit is used. This may result from the flexible motorised screen. Scanning the OHP picture with a second fixed camera is a good compromise for having optimal quality for the present audience and average quality for the Internet connections or VCR taping. The audio quality is extremely poor due to the use of the SUN Electret microphones. Volume adjustment has to be done manually using the vat tools.

5.2 Design of a Distributed Presentation Environment

At the University of Bonn we developed the Multi-Media-Lecture Room MMHS to support distributed teaching, discussion and presentation. The MMHS system integrates different audio and video media to support different tasks, like projection of Powerpoint slides, projection of video tapes and 35mm-slides, video tape duplication, titling and mixing as well as recording of a lecture and Internet tele-teaching or conferencing.

By evaluating other existing systems, some disadvantages could be avoided by using a different overall design. According to the different media types we detected some disadvantages of

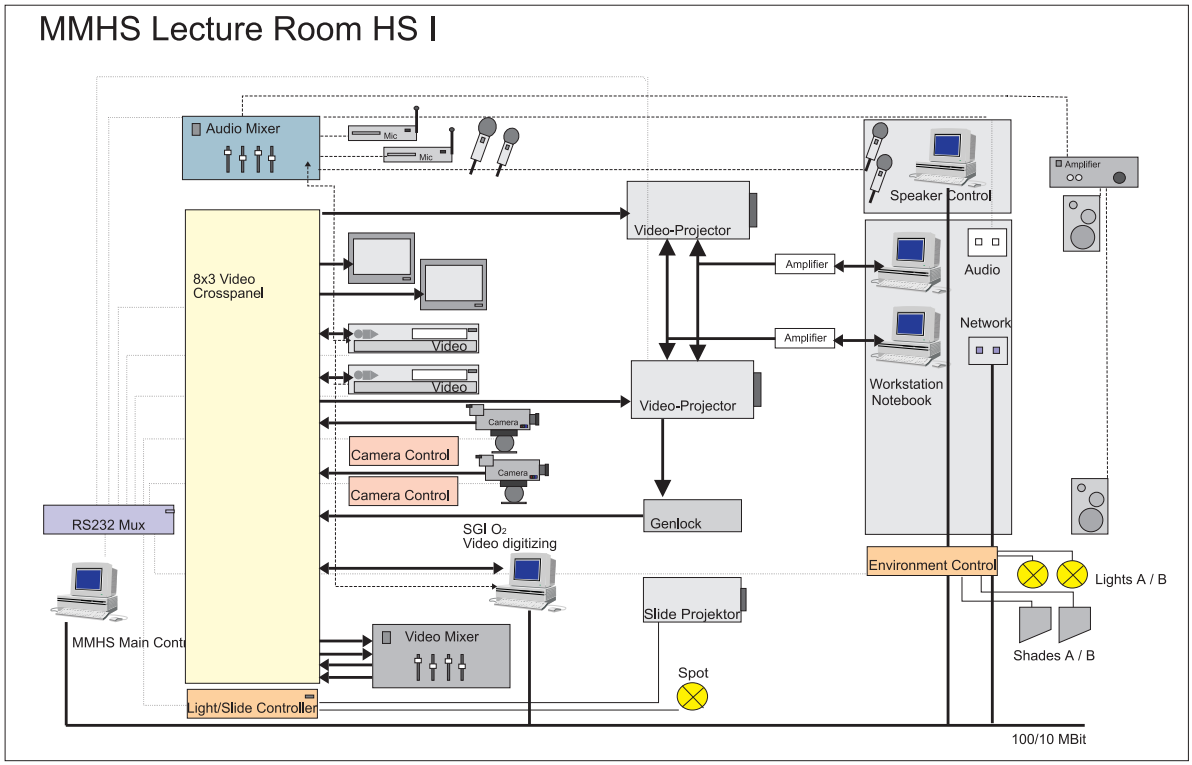

Figure 5.1: System Outline of MMHS

the existing solutions and can now give better solutions or technical hints to avoid these problems. This leads to our MMHS design. A main goal was a integration of all systems to allow a simple or remote controlled operation.

5.2.1 Supported Media

After interviewing different lecturers it became evident that tehe is the need to support different media to increase the quality of a lecture or perform distributed conferencing. The former section shows some facilities and their supported media types. Though resulting from the discussions on media quality, the supported media types are listed first:

- 35mm slides
- OHP projection
- RGB projection from workstations at least with 1024x768 pixels
- video projection
- NTCS/PAL VCR systems for recording and playback
- D/A video capturing
- audio amplification
- Internet conferencing, via SGI O2
- video cameras

While supporting these media the MMHS can be used for different purposes. As stated in the example application the distributed robotic experiment is a scenario for the use of the MMHS.

5.2.2 Video Media Quality

The idea of using different media in a MMHS is to achieve the best match of quality and presentation media. It is reasonable to distinguish (video) media by their resolution per square inch. A list of the different media types sorted by quality is:

- 35mm slides (approx 30 dpi)
- OHP slides, (up 24-48 dpi)
- RGB workstation output (approx 12-16 dpi)
- video output (7.2 dpi)
- video OHP system (7.2 dpi)

All numbers calculated for a 100" wide screen.

Though the resolution of OHP slides seems to be better at first glance, it is well known that the visual quality of 35mm slides is better than an OHP slide. This may result from the achieved contrast, which is not measured here, or the use of ink-jet printers, which do not produce images with an equal ink distribution.

As we have seen in other MMHS environments it means a loss of 4 times of picture quality, if we scan a slide by a video camera and project the video image on a video wall afterwards. The amount of quality loss during video projection is not considered. Thus, the conclusion that we have to choose the best projection system for each media leads to the following hardware requirements:

We need standard 35mm slide projectors, a standard OHP projection facility, a high quality RBG projection system for workstations and a video projection with the highest video quality.

5.2.3 Audio Media Quality

To achieve a high audio quality, we will have a look at microphones, audio mixing, volume and echo.

Figure 5.2: MMHS Main Control Console

In every room the audio waves produce an echo leading to feedback. So there is an essential need to have an echo compensation in the audio system. This can be done via echo compensators or by using special feedback free stage microphones.

The echo considered here is relatively small (approximately 24ms). Echo’s occurring on delays through long Internet transmissions has not been considered yet. Because of the irregular delay time of Internet transmission there has been no system which could compensate for

these echo's (0.5 to 2 s).

One solution for this problem is to have a mixing facility, selecting different channels for Internet transmission and for room audio amplification. Thus, it leads to different audio profiles in the MMHS and the transmitted signals. By the use of wireless portable microphones the requested amplification level can be low, resulting in a low amplification of room noise and echo. This is necessary for a echo-less transmission.

5.2.4 Components Used

In the following list we give a short overview oft the used components and their purpose and connection to the MMHS-system:

- two video cameras to capture the lecture room, audience and lecturer and OHP-slides; remote control of focus, zoom and light exposure
- Remote controllable Pan-Tilt-Units were used to control the position of the video cameras
- two VCR to playback and record video tapes in PAL and NTSC format
- 8x8 video-cross-panel to select and route video signals from cameras, Internet, VCR or Genlock to any output like projectors, VCR tape, control CRT or the Internet
- the D/A-capture device (Genlock) was installed to record Powerpoint slides on video tapes.
- a video mixing device to superimpose and mix different video streams to produce high-quality images
- two LCD video projection systems with 1024x768 pixel resolution provide video output for the audience, allowing digital zooming of video, PC and workstation images.
- A Unix SGI O2 workstation is used for the Internet connection using SDR, VIC, VAT and MRT-VR.
- A 14 channel audio mixing amplifier controls the different audio streams from microphones, VCR and Internet connections.
- environment control is done by two switching devices, allowing to control the lights and shades.
- A slide projector interface allows the control of slides and the focus of the 35mm slide projector.

5.3 User Interface Description

5.3.1 Media Control

Controlling of these different media is done via the main control PC and its 17" touch panel interface, the speakers touch panel interface and through an optional remote Java interface described in Appendix 9.2. All media is controlled via RS232 interface if possible, involving a 16 channel RS232 multiplexer for control. Because the different operators (main control, speaker control) have different intentions, the speaker's interface is reduced to control only the most important functions, while the main operator has full control. This leads to a complex interface. A two level design, consisting of a command interpreter and different user interfaces, allows an easy coupling of local and remote interfaces. The main command interpreter resides in the control PC, interpreting ASCII command lines like "CAMERA1 UP", "REKORDER1 PLAY" or "PROJECTOR1 VIDEO". These commands can be sent locally via strings or by remote using a TCP/IP connection. Results of the operations

are sent back via TCP/IP to the connected user interfaces. A complete list of the available commands is listed in table 9.9 and table 9.10.

Figure 5.3 shows the main control command screen.

The cross-panel in the middle dominates the interface. This control is used to route video inputs to video outputs. On the left we have listed the inputs (rows) while the outputs are located on top (columns). Pressing a button at the crossing of a column and a row routes the signal from the input row to the output column, indicated by a red cross.

To avoid occasional deletion of a routing, a double click on the cross-point can lock the column, which could be unlocked again by a double click. A locked column is surrounded by a red box.

Other controls on the main console are used to operate the shades and the lights. Because no feedback channel is provided the lights could only be toggled.

For the other devices a tab control interface was provided. Each tabular can be accessed by pressing the unit or the corresponding tab. Some of the tabs are explained in Section 5.6.1 in more detail.

For the speaker a reduced interface was designed and implemented on a control PC in the front of the lecture room. Both programs are connected via TCP/IP as shown in Figure 5.5. Instead or in addition to the speaker interface any other remote interface may be used.

The interface of the speakers console was designed to operate the most important functions of the MMHS system. As shown in Figure 5.4 the speaker can select the sources for the two video projection systems. Normally, he would use the source switch to route his notebook display to a projector to show some Powerpoint slides or a video tape. To operate the VCR, we have added the common buttons to control a tape recorder. Volume is adjusted by the volume sliders at the bottom of the screen. Lights and shades are operated with the control buttons on the right. The slide projector is operated with four buttons to step forward and reverse and to adjust the focus. If the user selects the slide projector as a source, the system automatically turns off the video projectors. Further automation is possible (e.g. shades and light operation). All buttons are presented on one screen only, so the user is able to learn the position of the buttons very fast. Used on a 14" screen, the buttons are big enough to be hit by the first try.

5.4 Internet Remote Control of MMHS

As stated earlier, the MMHS system is completely remote controllable. Messages are sent via TCP port to the main MMHS control software. The remote speaker software uses these features to control the most important functions of the MMHS system from the speaker console. A remote operator is able to help a speaker in controlling the system, if an Internet connection is used. Therefore a Java control software has been developed.

To control a session remotely via the Internet, at least one audio and video channel has to be transmitted and should be used for control. So, the Internet connection has to be established remotely by a rlogin to the Internet computer in the MMHS System.

Source code for a Java applet to allow a small remote control is given in Appendix 9.2. A screenshot of a Java application that controls the MMHS shows Figure 9.3. To control the MMHS ASCII strings are used as commands. Table 9.9 and 9.10 encloses a list of all commands available. The commands are sent to the MMHS control PC via TCP/IP on port 5001.

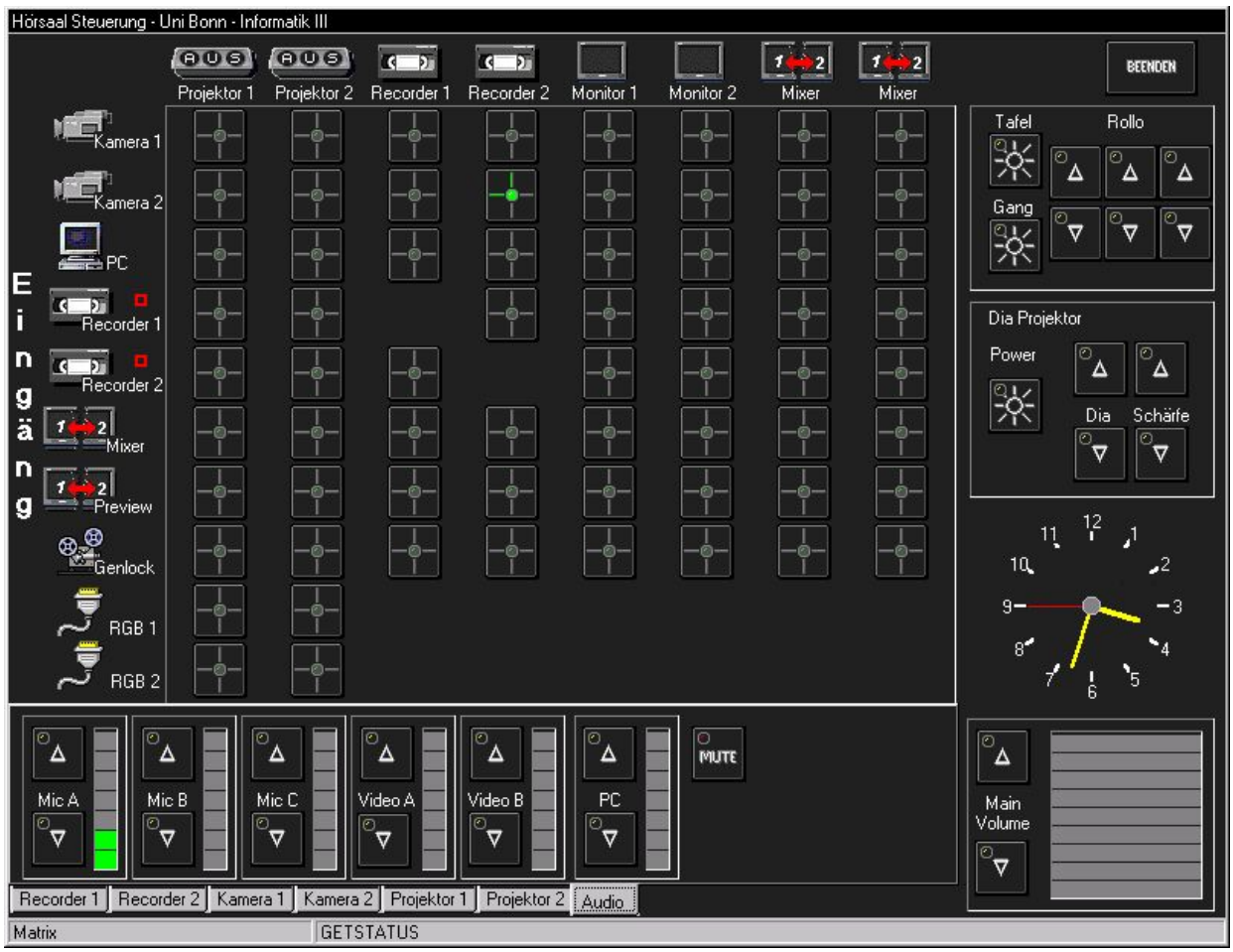

Figure 5.3: MMHS Main Control Console Screen

5.5 Usability Test

To evaluate a given system, we have to check its usefulness. Nielsen[Nie93] distinguishes “utility” and “usability” of a system. Nielsen describes “utility” as the question , whether a system can perform the tasks it was designed for or not, and defines “usability” as the question, how well users can use that functionality.

The term "usability" is defined as: "easy to learn, efficient to use, easy to remember, few errors and subjectively pleasing".

Because the system was designed to control the MMHS, and the test persons were able to perform the given tasks, the "utility" of the system is given. To achieve a high acceptance, we have to ensure the usability aspects of the MMHS.

The term "Learnability" means that "the system should be easy to learn so that the user can rapidly start getting some work done with the system".

"Efficiency" means that "the system should be efficient to use so that once the user has learned the system, a high level of productivity is possible".

"Memorability" means that "the system should be easy to remember so that the casual

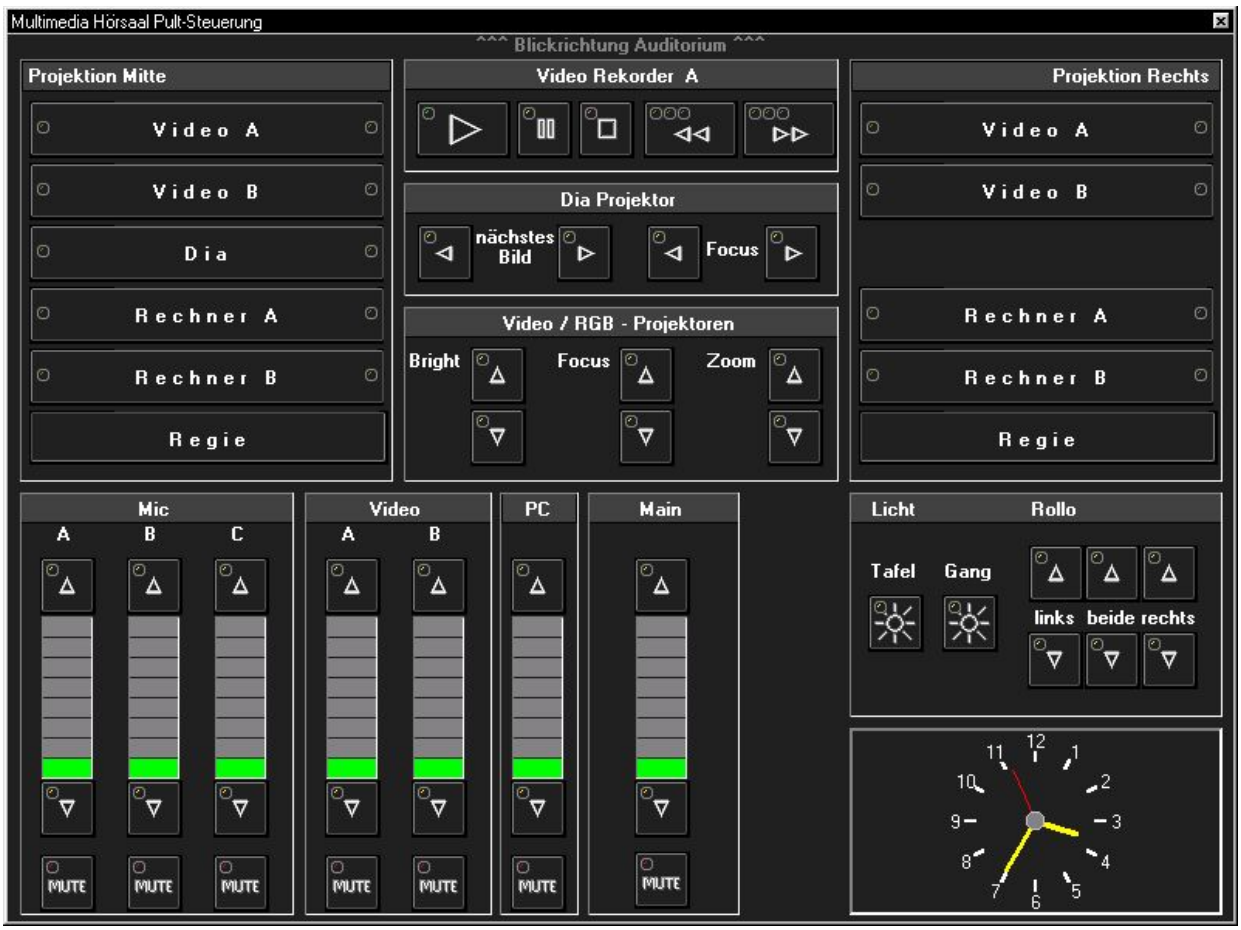

Figure 5.4: Speakers Control Interface

user is able to return ... without having to learn everything all over again".

"Few errors" mean that "the system should have a low error rate so that the user make few errors during the use of the system".

"Satisfaction" means that "the system should be pleasant to use so that users are subjectively satisfied when using it".

A first test was done using a test group of four computer experts which were unexperienced with the system. The test group was given a set of jobs to do within the MMHS. The test was designed to check different aspects of the usability term: “learnability” was checked by testing the time needed to perform a given task for the first time; “easy to learn” was tested in repetitive jobs and in measuring the different periods of times for the first and the second try to to complete these tasks. “Few errors” was tested by counting the buttons wrongly pressed; “subjectively pleasing” was tested by an interview of the test persons. By using the “thinking loud” method we could get information about misleading information; “easy to remember” was tested by doing the same test a few days later. As stated in [Nie93] a group of six persons is sufficient to get enough information. Questions were selected to perform different useful tasks and some questions were repeated. Additionally

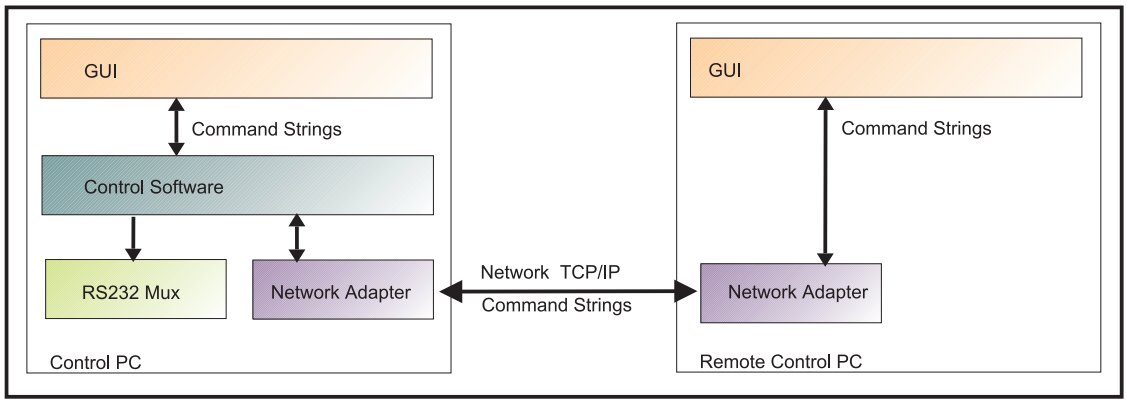

Figure 5.5: Internet Connection Between MMHS Control PC and other Remote Machine

the test persons had to answer some questions about the quality of the MMHS. We have used semantic differential scale lists for the questions to measure subjective satisfaction.

5.5.1 Questionnaire

To get the user's thoughts about the quality of the system, we used a questionnaire with the questions blow. At first, we wanted to obtain some information about the user's idea of system usability. We asked five questions and allowed a rating from 1 to 5 (with 1 for a good and 5 for a bad). To determine the user's thoughts about the reliability of the system, we asked questions about mistakes and the information the system returns.

As stated in [Nie93] the mean value of systems is rated 3.6 on a 1 to 6 rating, with 1 meaning "good" and 6 meaning "poor" quality. So, we have to subtract about 20% of the rating.

While recording the answers on tape and writing down some interesting remarks, the "thinking loud" gave us the interesting answers from the test persons listed in table 5.3 .

The next day we did the same test with three test persons who had listened to an introduction on the system an hour before. Each task had been explained in detail and the display had been shown in the MMHS. About one hour after the demonstration of the system four persons were tested. The questions were the same as the ones of the untrained test persons.

Usability		
positive Rating	Rating	negative Rating
easy to use	123456	not easy to use
complete	123456	incomplete
pleasing	123456	irritating
cooperative	123456	uncooperative
Mistakes are		
safe	123456	dangerous
System reaction		
fast	123456	slow
informative	123456	less informative

Table 5.1: Questionnaire Determines the Usability of the System

5.5.2 Results

The execution time of the 18 tasks was never longer than 15 seconds. Shortest execution time was approximately 1 second, average execution time 3 seconds. Average time for trained persons to execute a job for the first time was 5 seconds and 7 seconds for untrained persons.

Table 5.2 shows the answers of untrained per-

sons in the semantic differential scale.

Untrained Persons	
Question	Rating
easy to use	2222
complete	1112
pleasing	2312
cooperative	2322
safe	2214
fast	2424
informative	3324

Table 5.2: Answers of Untrained Persons

In table 5.3 we listed the comments the persons returned after they had tried to use the system. The value "times" states how often the test persons noted this comment down on the questionnaire. Some answers fit *n* different categories. Categories were derived from the comments.

Untrained Persons	
Notes	times
Video 2 Tasks	1
shutter problem	3
too much buttons	1
button reply too slow,un-apparent	1
light reflection problem	1
switch / push button problem	1
audio volume problem	3
slide video inconsistency	1
light slider distinguishing problem	1
RGB 2 select long time	2
important buttons bigger	1

Table 5.3: Problems of Untrained Persons, derived from "Thinking loud"

Table 5.4 gives the results of the trained persons. The comments of the trained persons, listed in the same categories used for untrained persons are shown in table 5.5.

Trained Persons	
Question	Rating
easy to use	212
complete	211
pleasing	221
cooperative	322
safe	211
fast	414
informative	222

Table 5.4: Answers of Trained Persons

Trained Persons	
Notes	times
shutter problem	3
button reply to slow,un-apparent	3
audio volume problem	2
buttons hit problem:	2
video 2 tasks	2
RGB 2 select long time	2
important buttons bigger	1
two buttons pressed same time:	1

Table 5.5: Problems of trained persons, derived from "Thinking loud"

5.5.3 Interpretation

Though we thought we had developed a good system, it needed only a few test persons to find the bugs we were already used to or tried to ignore with great effort. As developers we found the personal remarks from the test persons most helpful. Though these remarks were not always said during the test phase and so could not be directly related to the "thinking loud" method, personal interviews gave us the most detailed qualitative information.

The most interesting information for us was that five out of seven persons expected that the audio volume adjust buttons should have an auto repeat function (increase volume while pressed) or even that they should behave like sliders.

Also, six out of seven test persons tried a short press for the window shutter button at first. They expected the shutters to move down until windows are closed. Four out of seven persons had problems with slide and video projection inconsistency.

One person expected a panic button for emergency to stop all operation, another person had problems to interpret one left button as a "back" button for the slide projector. We did not recognize that colors were a great help for the identification of microphones or RGB ports.

No test person had a problem with the use of the VCR tape button design, which was expected because they are quite common.

5.5.4 Conclusions

The speaker's command interface was tested by using a questionnaire and the "thinking aloud" method developed by Nielsen [Nie93]. Personal interviews gave even more detailed qualitative information. Overall testing results showed that the different media systems involved in the MMHS are controllable even if the users are unexperienced. Some small changes are necessary to optimize the user interface. The interpretation led to the modification of the system though some problems are still remaining.

Problems

As a conclusion we state, that we did not expect so many problems with the buttons: Four out of seven persons stated that button reactions were too slow or un-apparent. Two persons mentioned problems with the buttons small size or said that the buttons showed inappropriate reactions. One person wished the buttons for important functions to be bigger and one person asked for different buttons to distinguish switches and push-buttons. Also we had a long response time to identify the correct notebook input port which was named RGB1 and RGB2. Test persons suggested to name the ports "notebook" and "workstation".

All major bugs were found by the first four test persons: One bug was that the VCR control did not work properly all the time. Another bug was that some of the users did not expect that the system waits for a press of the STOP and START button when the video was paused.

Because the adjustment of the touch pad is depending on the viewing angle it is not sure whether two test persons hit a buttons at the same position. So, it is necessary to have big buttons to ensure high hit rates.

After pressing the buttons a few times the test persons did not control the output of the projectors any more. It seemed that they trusted the system.

Only one person stated "I will test it.".

Re-design of the Interface

The hints of the test persons about the interface led to an enhanced design. Colors were not used any more to distinguish between ports or microphones. Buttons were placed further apart and separated by boxes. Different button styles were used for push-buttons and switches. The function for the window sliders was changed and a stop button added. The bug for slide back removed and the back button became smaller. The video "Pause" to "Play" transition was changed. Now, auto pause function is implemented when changing from or to video. The audio section was replaced by sliders which were set further apart. Some buttons were made bigger where possible. Active buttons were provided with bigger status indicators.

5.6 VR-LAB Hardware Control

To control the VR-LAB via PC or workstation, different hardware modules were needed to distribute control information to the connected hardware, like VCR and cameras. Wherever possible we used RS232 compatible hardware to reduce the amount of the necessary hardware to be built. Nevertheless, we had to develop extra controllers for environmental control, camera motion, camera control and audio amplification.

Figure 5.1 shows the main connections of the VR-Lab. The control PC (133 MHz Pentium) is located in the locker and running Windows95. The PC for the speaker control (133 MHz Pentium, Windows95) is located in the front desk. Both machines are equipped with a touch screen monitor. A description of the user interfaces can be found in section 5.3. As shown in Figure 5.1 the control PC is connected with the other units via an RS232 multiplexer. Figure 9.7 shows the wiring to the units. The video cross panel in the center acts as a main distributor for the video signals. As you can see the workstations (on the front desk) are connected via amplifiers to the projectors. The amplified signals are fed to the projectors and to optional control monitors. Video output from the cross-panel is also supplied to the projectors. Video input for the cross-panel comes from two VCRs and two cameras, the PC video output and a Genlock. The Genlock is the “way back” to the cross-panel to record computer images on video tape. A video mixer is connected to the cross-panel using two inputs and two outputs. So signals can be routed from a source to the mixer, and then to a sink, e.g. a monitor, VCR or a projector.

Audio signals are processed by the amplifier which is also connected to the RS232 multiplexer. The network connection between the front and the main control PC is used to transmit commands and state information, and can be used to control the system remotely because of the Internet link.

5.6.1 Hardware Development

In this section we will give an overview over the hardware developed for the MMHS environment. As stated before, all components are controlled via RS232 interface. Not all available hardware components have RS232 interfaces, so we had to develop some of them ourselves. To control the video cameras, we had to develop two camera control interfaces which control the pan-tilt head and the camera functions. The audio interface controls 14 analog channels and was implemented in an audio amplifier. Furthermore, we developed two six channel environmental control interfaces to control lights, shades and the slide projectors.

Development was done using the Microchip PIC series 16/17 microcontroller [Inc97]. Pic MCUs are designed for embedded system applications requiring high performance and user programmability. PIC MCUs are RISC type CPUs with Harvard architecture (separate buses for data and instructions). Most instructions are executed in one CPU cycle, each cycle taking 4 oscillator clock cycles, making timing calculations easy and resulting in a processing power of 2MIPS at 8MHz. We used the PICSTART-Plus development system for the development of the desired applications. For the proposed applications we used the one time programmable (OTP) version of the PIC 1657 with 3 output ports, of which two ports are 8-bit wide and one port is 5-bit wide. Port A was usually used for RS232 communication and status information display, while port C was used for data output.

For RS232 communication the MCU had to be equipped with a MAX232 RS-232 line driver circuit. The environment controller and the camera controllers use electronic solid state relays to switch the AC current for lights,

shades or the camera motors. The infrared controller for the video cameras can be directly driven by the MCU. The proposed circuit layout for the 5 controllers employed in the MMHS and the control programs are shown in the Appendix.

Camera Controller

The camera controller was built to drive a VPL pan-tilt head and a Sony camera TR825E or equivalent. The camera controller allows the control of the camera functions zoom, focus and over exposure for back-light situations and the drive of the pan-tilt motors. The infrared LED module transmits an IR command three times to the camera module. Due to timing reasons the controller can not process commands during transmission time. The motor controller switches the outputs permanently, so that one command is sufficient to enable motion. If the camera should stop a movement, the corresponding bit has to be cleared again. A touch-pad “button down” event should be mapped to a “motor on” command, while a “button release” event should be mapped to send a “motor stop” command.

Data Transmission Data transmission is realized via RS232 interface at 9600 baud, 8 bit, no parity and 1 stop bit. Commands are 1 byte long and no start or stop byte is required. The controller uses RX and grounding only. TX transmits an ACK, if a signal can be received. The control lines CTS and RTS are not used. The controller consists of two parts: the motor controller and the IR controller. To distinguish between both units, the MSB bit 7 is used. Control indicator LEDs are used to indicate power (5V) and “circuit alive” (ready). A data LED is lit when data can be received via the RS232 line. The data LED is used to signalize a visible equivalent of the transmitted IR signal.

Figure 5.6: Sony Camera mounted on Pan-Tilt Head

Timing Considerations The unit should be powered off, before powering off a connected PC. While the PC grounds RX when powering down, a valid signal may be received by the controller forcing the pan/tilt head to move. The sending of an IR signal takes approximately 60ms whilst a motor control command can be processed immediately after being received. If more than one command has to be processed at a time, controller commands should not be sent more often than every 60ms to avoid unpredictable behavior of the controller (a command could be partially overwritten).

Infrared Controller IR transmission requires a 14 bit serial transmission of a 40kHz modulated IR beam. When transmitting IR commands a 7 bit code is transmitted at first, and a 7 bit signature is transmitted afterwards. To increase user’s convenience the commands are completely coded in the controller software so the user has to select available codes from the table. Other units, with different signatures, can be supported by simply exchanging the sign code in the controller software. Other signa-

Figure 5.7: Camera Operation Panel

tures can be used to control the built-in recorder. Command codes for the camera controller are listed in table 9.8.

Hints For safe operation it is required to set only two of the available motor control bits at the same time and not to set two opposite bits (e.g. up and down) at the same time.

Environment Controller

The environmental controller is a RS232 controlled circuit, with 6 solid state outputs. Two of these controllers were used in the MMHS.

Data Transmission The environmental controller is operated at 9600 Baud, 8bit and no parity. Each command is one byte long. Only the RX and GND lines are used. RTS and CTS are not used. Each bit corresponds to an appropriate output. So commands are very simple.

Control LEDS There are three LEDs mounted at the controller to indicate the system status. The +5V LED indicates a correct power supply, while the flashing READY LED indicates the correct operation of the MCU. The red DATA LED indicates data transmission on the RS323 line.

Bits:	function
11xx xxxx	Output
1111 1111	all outputs inactive
1111 1110	output 0 active
1111 1100	output 0 and 1 active

Table 5.6: Control Codes for the Environmental Controller

Outputs The outputs of the solid state relays (SSR) do not drive 240V AV directly. The output of the SSR is feed to relays mounted by an electrican. Each relay can drive 1500 W. The number of concurrently activated relays should be less than 10, according to the power source limitations of 200mA and to each relay consuming approximately 20mA.

Timing Considerations The controller does not support interrupts, so there is a delay after a command has been received, and then the controller is ready for the next reception. Relay outputs are buffered, allowing only one switch operation per second.

The output-unit assignments of the environmental controllers can be found in table 9.5 and 9.6. Table 9.5 describes the pin assignment of the environmental controllers.

The slide projector interface is special in the way that the direction of the slide transport has to be set before the transport is enabled. The mechanism to change polarity of the transport

Figure 5.8: Screenshot of the Environmental Controller Slide Projector Interface

motor was developed by using four relays mounted in the locker.

Audio Controller

We used a Samson mixer S602 as active mixer and amplifier to mix and amplify up to 14 audio channels. The RS232 controller is used to control the volume of the mixer remotely.

Hardware Changes Due to space limitations in the amplifier, only the volume potentiometers were replaced by electronic versions. The external input socket for Audio channel six was disabled and used to connect the RS232 controller. The MCU adjusts the volume for 14 channels. A table below gives the assignment of channels to the potentiometer of the amplifier.

The Samson S602 operates as a six channel amplifier with two independent output lines. The two output lines were used to feed the Internet-line and the local MMHS environment. The other two channels are used for the main volume and reverb effects.

The circuit uses a PIC 1657 and 14 DA converter, built-in the amplifier.

Control The audio controller is operated at 9600 Baud, 8bit and no parity. Each command is one byte long. Only the RX and GND lines are used. RTS and CTS are not used. The external power supply has to be switched on before and switched off after the main power of the amplifier. Otherwise, the controller may switch the volume to full power.

When initialized the controller operates a reset, which means setting all outputs to minimum value. If the controller is not powered up all DA converters are set to low, which results in maximum volume. All DA converters have a resolution of 100 steps. They can be operated in a single step or 5 step mode. The least significant bits select the channel while the four MSB select the function.

The available control codes can be found in table 9.3. The assignment of audio channels to control channels is listed in table 9.4.

5.6.2 Third Party Hardware

Wiring

Most of the video wiring has to be accomplished using five wires of RG-58 video cables, because of the long distances to the projectors. The Genlock device is used to feed back the video signal to the cross-panel, which was mounted in the media locker requiring a five wire video line to the locker, allowing further extensions. Remote control of the lights and shades requires several power lines and a switchable power-line for the whole system to be booted by a key turn.

To control all components of the MMHS the Moxa RS232 multiplexer is operated by the main control PC at Com port 3, expanding the

number of com-ports by 16. Table 5.7 shows the assignment of RS232 Com-ports to the connected components. All components of the MMHS system are operated via RS232 to establish a common interface for all devices. We used a common CAT 5 TP cable for the serial links. Figure 5.1 shows the overall wiring diagram, whereas video wiring is shown in Figure 9.6 and control wiring in Figure 9.7.

NEC LCD Projectors

The LCD projectors of type NEC 1000 LT are RS232 remotely operable, which is much simpler than emulating an infrared controller. The projector offers many commands, normally accessed by the infrared remote control. This is difficult for unexperienced users, in particular, when two of the projectors have to be operated at the same time and the IR signals could interfere. The implemented touch-panel makes operation very easy. We did not implement all of the functions in the remote interface, because these may be used by experienced operators only, and the interface should be kept as simple as possible. The projector manual contains all available commands. A part of the command set of the projector is listed in the table 9.2.

A remarkable feature, digital zoom, was one of the functions implemented in the control panel. Digital zoom allows to select an area of interest of the image and to zoom into it. The interface was designed equal to the camera positioning and is very easy to use. Different buttons control brightness, contrast, zoom and focus of the projectors. Input selection is done via the cross-panel on the touch-screen. Figure 5.9 shows the user interface for the projector available at the main control console using the projector tabs.

Crosspanel

The Kramer 8x8 video/audio cross-panel is an 8x8 matrix to distribute eight video inputs to eight different outputs. It is controlled by RS232 at 9600 Baud, 8-bit and no parity. Communication with the unit is done using a 4 byte command. The first byte contains the unit address (usually 0), an output and an input byte and a CR (13) to terminate the control sequence. The output and input bytes have to have a set MSB bit.

The current switching state of the unit can be determined by sending a 4 byte command which selects the channel of interest. The unit returns a byte indicating the value of the channel. Because every output can only have one input, the connection relation for each output is 1 out of 8, so the return string gives the input i for a given output n.

The query command for an output channel i is 4 byte long and starts with the unit identifier (0), a command 10 (10+0x80), the channel of interest (i+0x80) and a CR (13) to terminate the sequence. In return, the unit gives the selected input channel.

RS232 Multiplexer

The MOXA RS232 port multiplexer supplies 16 RS232 ports on a normal PC. It operates under Win95, WinNT and Linux. Enumeration of the added RS232 ports starts at COM3.

VCR Phillips

The Phillips VCR 9630 can be controlled via RS232 interface. Communication is done at 9600 baud, 8-bit and no parity. VCR 9630 accepts several commands to control the player functions, which are listed in table 9.2. When operating the Phillips VCR from remote the time between two commands should be greater than one second. Especially the status

Figure 5.9: Screenshot of the Video Projector Control Interface

Connector	ComPortControl	Unit
1	COM03	Kramer 8x8 video cross-panel
2	COM04	unused
3	COM05	camera1 (center)
4	COM06	Camera2 (window)
5	COM07	VCR 1
6	COM08	VCR 2
7	COM09	projector center
8	COM10	projector window
9	COM11	environmental controller back
10	COM12	environmental controller front
11	COM13	audio controller

Table 5.7: Assignment of Components to MOXA RS232 Multiplexer

command ST can block the recorder. The recorder can not be operated manually when connected to a PC and status calls were made. Time-code information can be returned by the recorder only if the tape was recorded with it or an equivalent machine. A full command set and a description of all command codes is available at Phillips.

Chapter 6

3D-Projection

Video systems normally provide 2 dimensional images of the 3 dimensional world. Especially in the described environment 3D projection would improve the immersive effect of three dimensional worlds. To achieve an immersive 3D impression, a stereoscopic projection system could be used. For a stereoscopic impression we have to calculate 2 images, representing the left and right eyes' view of a human being. The inter-ocular distance is normally about 6 to 7 cm. Given a focal point of interest in the 3D world the two camera positions and viewpoints can be easily calculated and two images can be generated by the underlying graphic hardware.

6.1 Stereoscopic Projection

A main problem is to feed the two different 2D images of the 3D world to the two eyes of the audience. Conventional systems are separated in active or passive projection systems.

6.1.1 Active Projection Systems

Active projection systems use a time-division-multiplex technique and shutter glasses to separate the two video channels, as illustrated in Figure 6.2, image B2. The two images are switched by frame or by line and are presented by one projector. Synchronous to the switching event the corresponding shutter of the right or left eye is closed or opened. So the user sees only one image at a time. Normally, this technique is used with fast CRT projectors or CRT monitors, which are operated at 120Hz, resulting in a 60Hz frame rate for a single eye. The switching of the shutter glasses is very efficient, resulting in a contrast ration of almost 90

6.1.2 Passive Projection Systems

The passive stereo projection uses polarized glasses and normally two projector images to generate a stereo video image, illustrated in Figure 6.2, image B1. The two images are projected in parallel on the same screen, using two projectors, which can be operated at a standard frame rate of 60 or 80 Hz. The images are separated by the use of polarized filters in front of the projector lenses and by wearing polarized glasses. When used with a screen of high quality, the resulting images are separated as well as when using shutter glasses. The image is brighter than the image using active stereo because two projectors were employed producing the image, and the brightness is reduced only by the polarization filters of the projectors. (Shutter glasses are polarization based, so they have the same dimming effect than polarised glasses, and can be omitted by this calculation.)

6.1.3 Discussion

The known systems suffer from some disadvantages. On the one hand, active stereo needs expensive shutter glasses [1] and projectors with high frame rates to generate a stable image. Normally, a CRT projector is used at 120Hz consuming an extensive amount of video bandwidth and leading to expensive projectors, expecially when bright images and high resolution have to be produced.

The passive system needs at least two directly accessible graphic ports, which are available on very expensive systems like a SGI Onyx. On the other hand, two images have to be projected at the same time and have to match exactly on the screen. For this reason expensive CRT projectors are used, which allow convergence adjustment for many image fields. Normally, these projectors have lens diameters of 10 to 20 cm leading to very expensive polarized filters.

6.2 An alternative Setup for 3D Projection

To achieve a good but less expensive 3D-effect for a big audience, only the passive stereo mode can be used, because of the available cheap [2] glasses. When using a LCD instead of CRT projectors, cheap polarization filters can be used for the projectors. Normally CRT projectors have three, LCD projectors just one lens, which could save even more money. Because of their small size, LCD projectors could be mounted almost on the same optical axis. When using a tele-lens the parallax effect is very small. New projectors come with a keystone correction so that the parallax could be eliminated very easily, although this setting can be used with machines with two graphic ports only.

Using a standard PC with one video output is not possible with the solution presented above.

6.2.1 The Synchonisation-Separation-Circuit

While using a Synchronisation-Separation-Circuit the images can be separated to two channels and feed to the projectors making use of their built-in video frame buffers, illustrated in Figure 6.1, image C1 . Figure 6.5 illustrates the function of the Sync-Separation-Circuit. Using the frame buffers of the projectors leads to a dramatic decrease of the video bandwidth needed to feed the projectors. Theoretically the images have to be updated only, when screen content is changing, e.g. when an object or the camera is moved. Practically the projectors were operated at 50Hz to 60Hz. Remember, that LCD and DLP projectors are operated at 60-120Hz *internally* producing a stable image independent from the input frequencies.

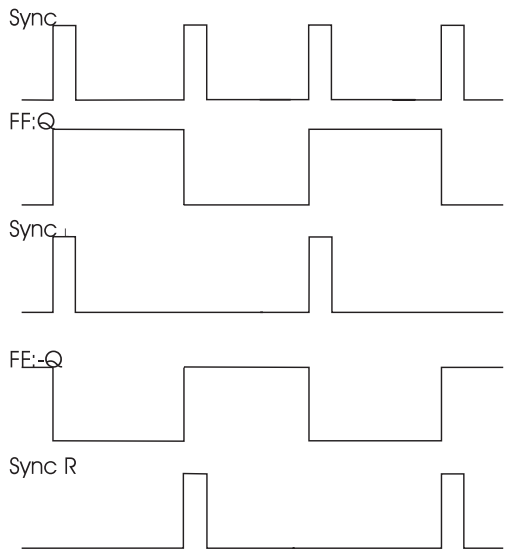

Figure 6.5: Signal Trace of Sync-Separation-Circuit

Figure 6.5 shows the function of the sync

[1] Although recently ELSA launched a set of very cheap shutter glasses

[2] About 0.5 Euro a glass

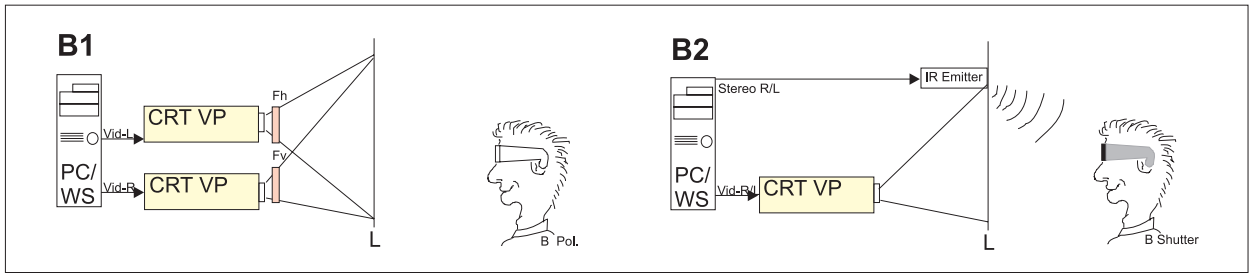

Figure 6.1: Passive and Active Stereo Projection Systems, State of the Art

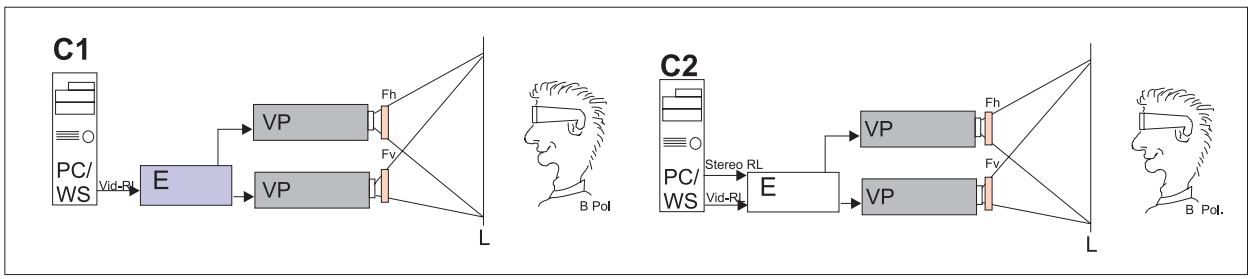

Figure 6.2: Stereo Projection Systems, proposed passive setup

separator circuit in more detail. The V-sync pulses of a standard PC display are fed in a flip-flop and producing the output FF-Q and FF_-Q. AND'ing these signals with the sync signal produces an alternating sync signal. Feeding these two signals to two different video outputs leads to 2 pictures synchronized every half frame. One picture synchronized on every even frame, the other on every odd frame. One screen output is showing the even frames on top, the other the odd frames. For example an image of 800x600 pixels (left and right frames) is now split into two pictures of 800x1200 pixel. Using a described projector with an internal frame buffer memory, only the top 600 lines of the frame buffer can be displayed. Using a 800x600 display, leads to 600 lines displayed though 1200 lines are delivered. So a simple LCD or DLP projector can be used to display stereo images.

The sync separator can be used with computers with or without a stereo emitter output. In the case a stereo emitter out exists (see Figure 6.1 Image C2) it can be used instead of the flip flop which guarantees the correspondence between right image with right eye and left image with left eye.

Alternative settings for 3D projections

Connecting the IR emitter for shutter glasses with the sync-out of an LCD or DLP video projector allows to use these cheap projectors with shutter glasses, too. Although it is not very difficult to develop a projector which supports a sync-out there is no such projector available at the moment. DLP projectors could be easily modified to generate a sync-out signal due to their optomechanical construction. With a sync out signal and the sync separator even cheap PCs could be used to project stereoscopic images, see Figures 6.3, image C3 and 6.3, image

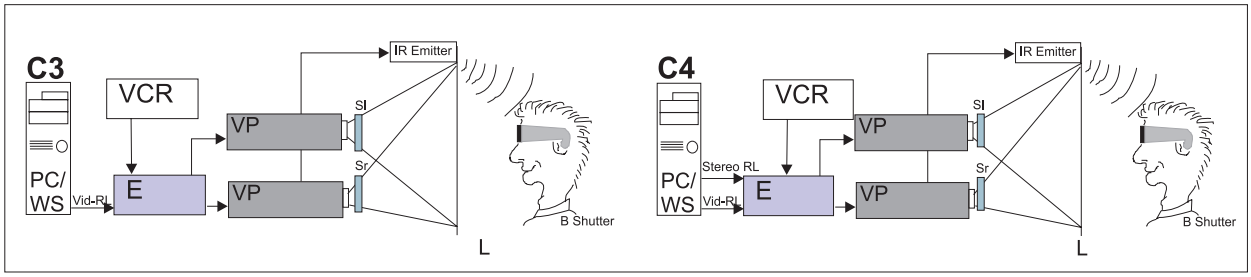

Figure 6.3: Stereo Projection Systems, proposed active setup with two projectors and shutter system

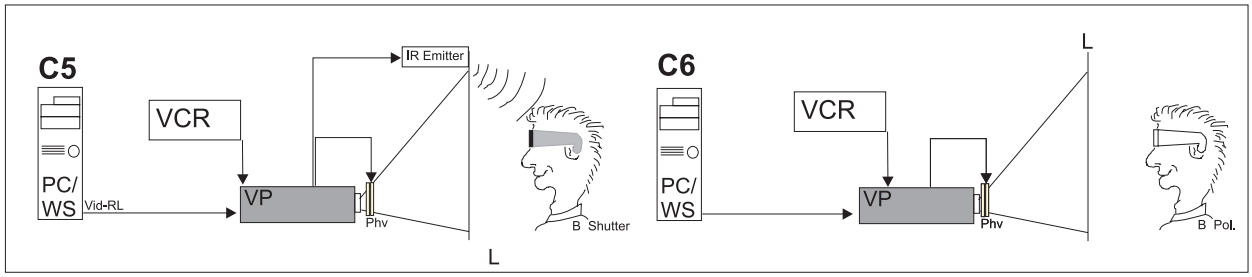

Figure 6.4: Stereo Projection Systems, proposed single projector setup, active with shutter and passive with active polarizer

C4. Additionally the projectors would have to be equipped with a shutter device S for the right and left channel. As an electronic shutter the LC shutter glasses of standard shutter glasses could be used as shown in Figure 6.7, Image E.

Figure 6.4,image C5 shows a setting with just one projector used for the generation of stereoscopic images with shutter glasses. This setting can be used to see standard video in 3D, based on the delay effect between right and left eye images.

To use the only one projector with passive stereo projection, this projector has to be equipped with a polarization switching mechanism. Figure 6.7,images D,E and F show three possible implementations of this mechanism. In Figure D a filter wheel is placed into the light beam, rotating with half the frame rate, leading to every even frame polarized horizontal and every odd frame vertical. The same technique is used in Figure 6.7,image F where the filter wheel is rotating around the projector, allowing the use of very cheap filter material, slow rotation when many filters are used and an easy setting of the mechanism. Setting E shows an electronic version of the polarisation mechanism using two shutter elements, which can also be used as an electronic shutter mechanism, too. Implementing an electronic shutter in the projector allows the system to be used as a cheap stand alone stereo projection system.

6.2.2 Alignment of Polarization Filters

When setting up a passive projection system the orientation of the glasses and the polarization filters of the projectors have to be synchronized. Especially, when the user in front of the projection screen is turning his head a misalignment may occur. Avoiding misalignments can

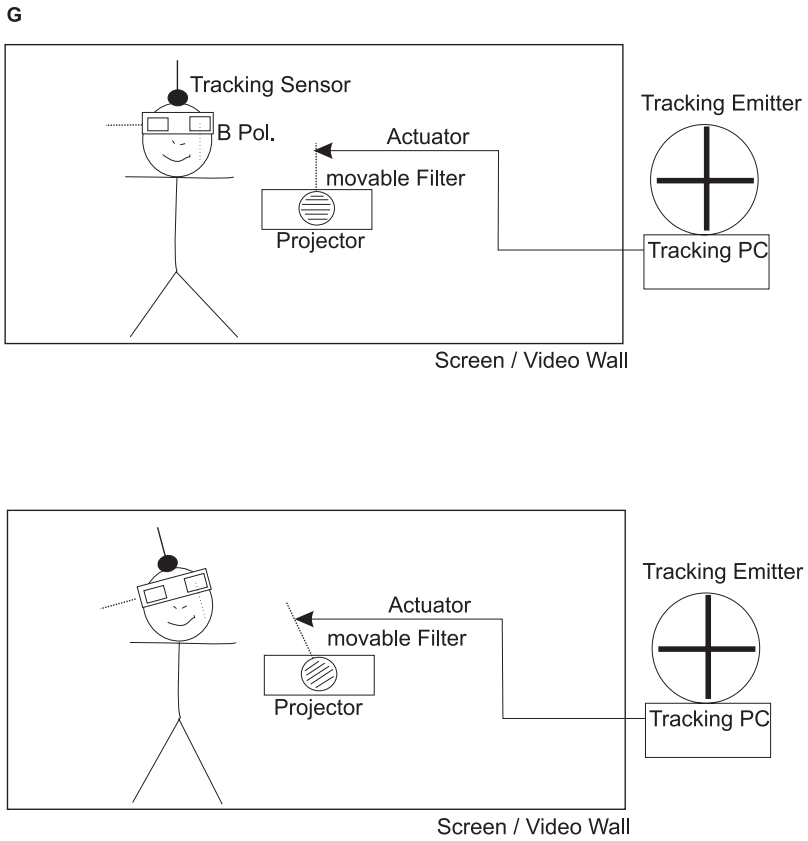

Figure 6.6: Tracking Setup for Passive Stereo Projection, Head Tracking Mechanism Used to Align Polarization Filters

be managed by using a standard tracking device, which is normally part of the setting, and calculate the head position, which is normally done to calculate the simulation camera. This position is used to change the rotation of the polarizing filters within or mounted at the projector. A simple stepping motor device can be used to align the polarizers according to the head position. Using this correction method is very important in a CAVE [3]. When using a CAVE different axes of the head movement have to be taken into account and should be used to rotate according filters for top/bottom front and side projectors. How to set up a back projection system is described in Figure 6.7, image G. When a head movement is measured by the tracking device, the alignment of the polarization filter is changed via the actuator mounted at the projector.

6.3 Future Enhancements

Using the proposed communication infrastructure a scene can be rendered by several computers in a local network. Using the MRT-VR's "what you see is what I see" function, all machines render the same image. If we added a feature, which holds the eye point but changes the look-at point relatively to the actual look-

[3] A CAVE is a small room, where up to 5 walls are used as projection screens, leading to a good immersive effect

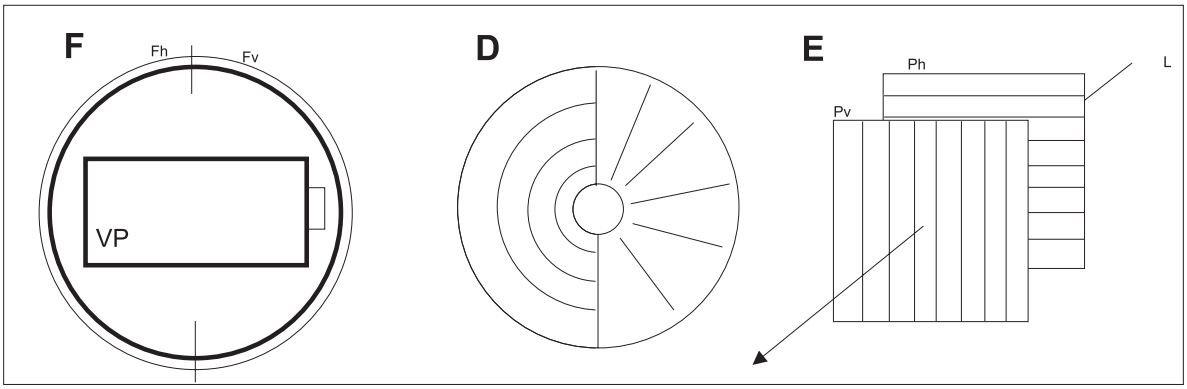

Figure 6.7: Polarizers for Passive Stereo Projection

at point (to the standard coordinate axes, front, up, left right and down) we could use a local rendering cluster to produce the images needed for multi-projection systems like a CAVE. This leads to a much cheaper design than a conventional multi-pipe Onyx approach.

MRT-VR may be modified additionally to select the field of view in a specific direction, to enable multi- machine rendering of one image, (e.g. 2x2) so that many cheap machines can be used to render a single picture. Multi-projector systems may be used to combine such images. This is simple done by using cheap LCD or DLP Projection systems, especially when using a back projection system.

Chapter 7

Sample Experiments

In this chapter some experiments, performed with the MRT-VR and the VR-Lab are described in more detail. Some experiments were used to test the environment and the software. The appropriate results are presented in this chapter.

7.1 Sample Experiment: Robotic Simulation with the RTL

In cooperation with the robotic tele laboratory (RTL) at the University of Bonn we realized a virtual robot experiment, which visualized a robot moving through the institute [Sch98]. Goal of the experiment was to show, that the robots movements could be controlled much better when distributed via the Internet and visualized by MRT-VR than by a bandwith consuming video transmission.

7.1.1 The Robotic Tele Labor (RTL) System

The robotic Tele Laboratory System RTL [CBS98] is designed as an experimentation platform which permits distributed researchers to carry out experiments over the Internet with an autonomous mobile robot [TBB*98]. Because of the varying and sometimes very low bandwidth of the Internet, RTL relies on 3D graphic visualizations of the robot and the laboratory environment instead of using video transmissions. In addition to lower bandwidth requirements graphic visualizations offer more flexible inspection possibilities than video transmission. Experimenters can choose arbitrary viewpoints in the virtual scene, while they are restricted to the viewpoints of few statically fixed cameras when using video transmission.

RTL employs MRT-VR as the user interface for the tele-experimenters. On the one hand, RTL benefits from the advanced virtual reality navigation and inspection mechanisms of MRT-VR, without having to implement its own visualization component. On the other hand RTL, adds an other inspection technique, based on automated viewpoint switching, which makes the observation of the robot in its environment easy. Furthermore, MRT-VR is enhanced by RTL with a simulation based dead reckoning component enabling it to perform smooth animation of software controlled 3D objects even when large Internet transmission delays occur. RTL has a client-server architecture and is currently being implemented for our mobile robot RHINO, an RWI B21 synchro-drive robot equipped with 2 laser range finders, a ring of 24 ultrasonic sensors as well as with several tactile and IR sensors (see Figure 7.1). The robot control system of RHINO consists of several software modules,

Figure 7.1: The RWI B21 Robot RHINO.

each performing a distinct part of the robot control task, like collision avoidance, robot localisation, path-planning and task-planning [TBB*98]. The RHINO project focuses on the development of a flexible service robot platform which can be used, for example, as a delivery robot in office environments as well as a mobile information agent. The server of the RTL system is a module of the RHINO system and a member of a MRT-VR session at the same time. As a module of the RHINO system it receives the current position of the robot as well as the current plan of the robot's future actions from the responsible modules of the RHINO system. As a MRT-VR session member, it animates the robot's avatar in the virtual scene based on the information obtained, and decides on its own viewpoint to take inside the scene depending on the robot's current position.

A client of the RTL system is an MRT-VR client enhanced with a special robot simulation component. The RTL client receives MRT-VR messages augmented with robot odometry and goal information from the server. This additional information is filtered out by the robot simulator inside the client. This simulator predicts the behavior of the real robot and produces intermediate MRT-VR update messages when transmission delays occur. It simulates the odometry and the proximity sensors of the robot, and employs a replication of the robot's path planning and collision avoidance facilities to achieve a reliable prediction of the real robot's actions. The accuracy of RHINOs localisation module in combination with the reliability of the predictive simulation of RTL ensures that RTL always presents a nearly exact representation of the real situation in the laboratory (see Figure 7.2).

Automatic Camera Selection in RTL

The RTL-server acts as an MRT-VR session member. As such it is able to define its own

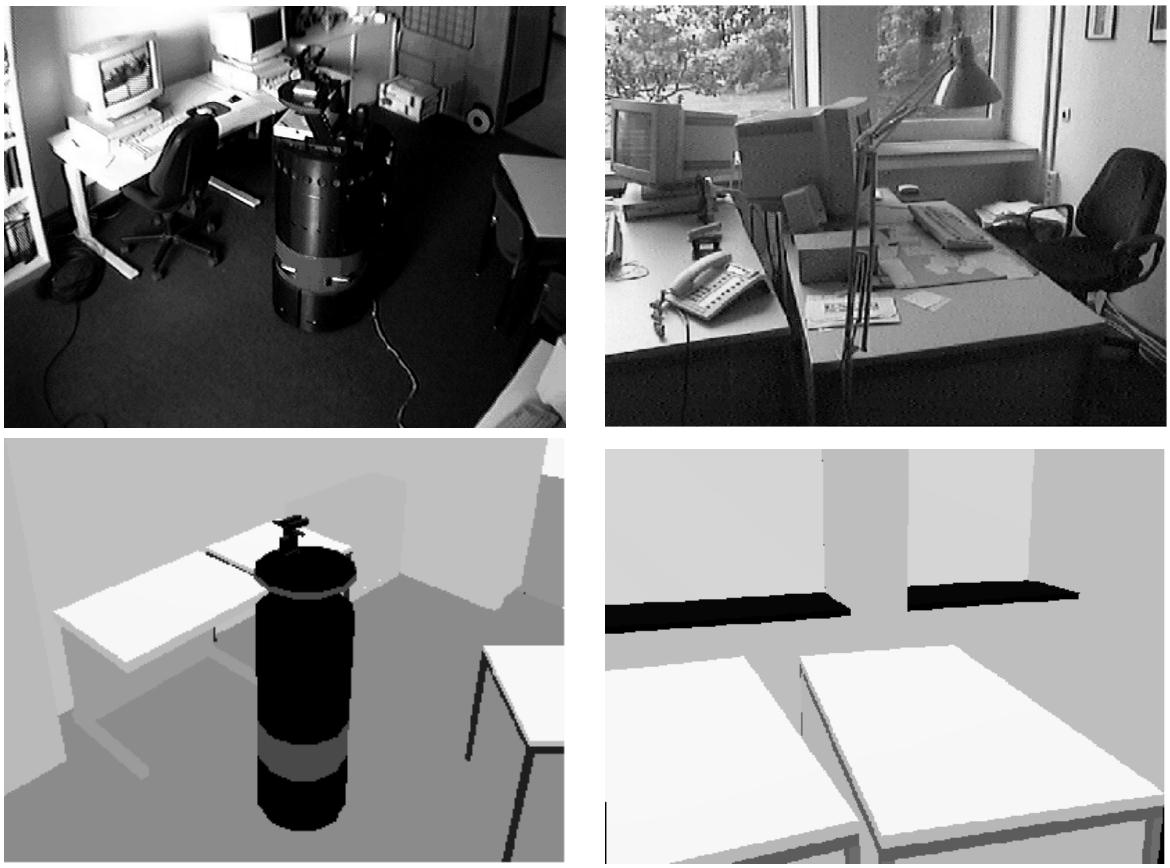

Figure 7.2: RTL Permits 3D Visualization of Real Scenarios. The two pictures on the left show a real and a synthetic picture of RHINO in an office. The two pictures on the right show a real and a computed view through RHINO's Camera.

camera position. RTL exploits this ability to automatically change the experimenter's viewpoint during an experiment.

The need for such a mechanism arises quiet naturally during the design phase of a tele-experiment. In most cases, the experimenter has a clear idea from which viewpoint he wants to follow a critical phase of an experiment, for example how elegant and fast the robot manages to detect and pick up an object from a desk. Changing viewpoint manually may be too slow in these occasions, as the navigation of a virtual camera is still a difficult job.

RTL implements a simple automatic switching mechanism, which is easy to specify during the experiment design phase. The mechanism is based on defining regions of interest, each of them linked to a static viewpoint. A region of interest (ROF) is a volume in (x, y, Θ)-space, the space of robot positions (x, y) including the heading Θ.

RTL decides on the current viewpoint depending on the robot's position based on an adjacency relation between ROF's. It decides on a new viewpoint every time the robot leaves an ROF *A*. The virtual camera is then switched to the viewpoint defined by the ROF *B*, which has been entered and which is adjacent to *A*. However, the user is not forced to specify a adjacency relation, in which for each ROF for all possible situations a unique adjacent ROF exists. RTL solves for ambiguities and omissions heuristi-

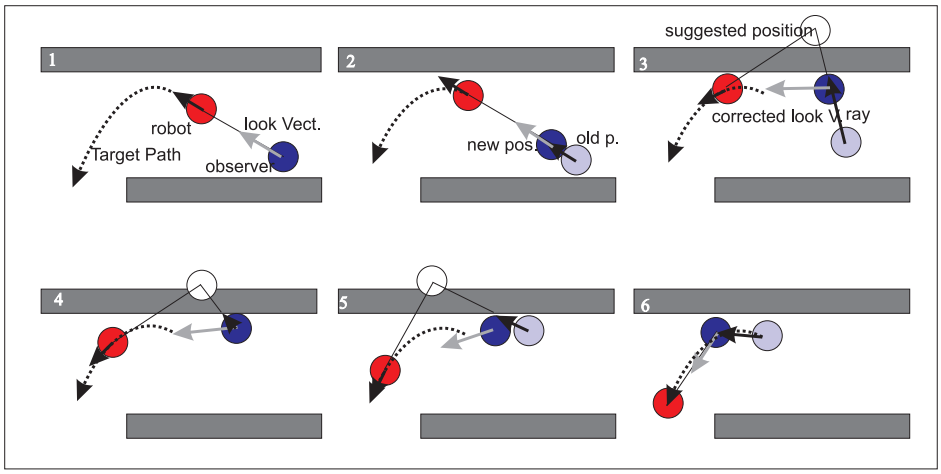

Figure 7.3: A virtual camera following the robot. The collision with the wall is detected and the camera is set to an appropriate position, in the line of sight to the robot

cally:

1. If an ROF has several adjacent ROFs for a robot's position, RTL nondeterministically chooses one of them.

2. If an ROF has no adjacent ROF for a robot's position, RTL nondeterministically chooses one the ROFs, in which the robot's position is situated.

3. If no ROF exists at the robot's position, RTL does not change its viewpoint.

4. Initially, a ROF is chosen according to rules 2 and 3.

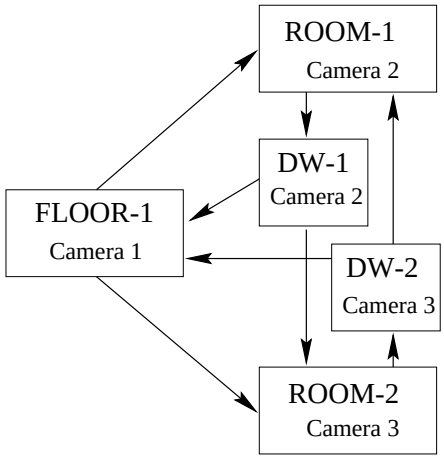

Figure 7.4: Transition Scheme for the ROF Example

Figure 7.6 displays an example environment covered by 5 ROFs. The directed graph in Figure 7.4 visualizes the adjacency relation of this example, which does not depend on the robot's heading to keeping it simple. Note that ROFs may overlap. In fact, the ROFs DW-1 and DW-2 in Figure 7.6 denote the same (x, y, Θ)-volume but they are linked to different viewpoints. RTL keeps the viewpoint of camera 2 when the robot leaves ROOM-1 as long as the robot moves toeards the doorway DW-1 and RTL keeps the viewpoint of camera 3 when the robot leaves ROOM-2 as long as it moves towards DW-2.

Theoretically, this technique allows to approximate any viewpoint control which is solely based on the robot's position. Even camera movements depending on the robot's trajectory can be simulated by using a large number of small ROF's covering the trajectory and linked in sequence.

Although the adjacency relation for such a complex camera control might be generated au-

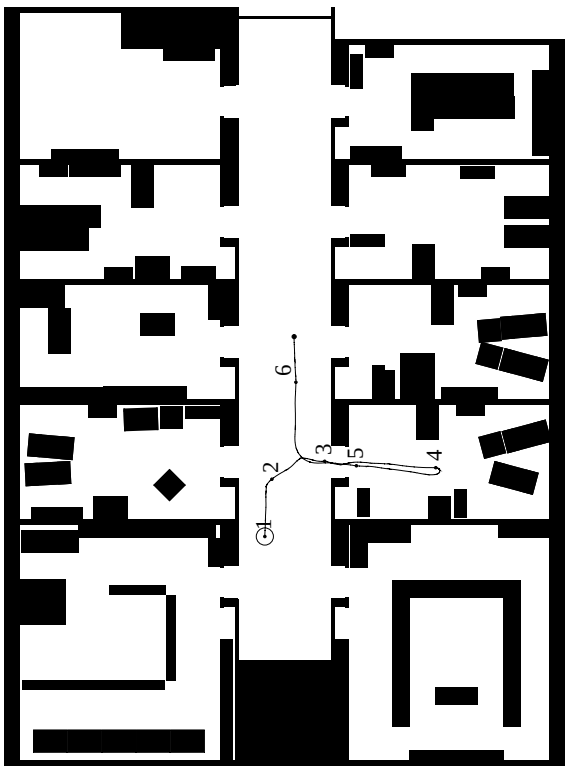

Figure 7.5: Path of the Robot During the Experiment

tomatically, it will unavoidably lead to an explosion of the number of ROFs used. For the same reason, it is not feasible to generalize the region of interest concept to more degrees of freedom like manipulator positions. For such complex tasks it seems more appropriate to use a rule based approach for camera control.

However, tele-experiments usually require only a small number of ROFs which are linked by a simple adjacency relation (comparable to the situation in Figure 7.6). So, the specification of the camera switching behavior is one of the easier tasks during the experiment preparation. Defining a consistent set of rules seems far more difficult. Furthermore, the ROF approach is very efficient. It only requires one (x, y, Θ)-volume inside check after each robot position update plus the finding of a new ROF after the robot left the last one. This takes $O(\log N)$ time in the worst case, where N is the number of ROFs present, for practical adjacency relations it is even faster.

Tele-Experiments

Currently, RTL is mainly used for the evaluation of experiments concerning robot navigation strategies in our own office environment. Questions of interest during these experiments include details of the behavior of the navigation system of the robot like: “How elegantly is the robot able to enter or leave rooms?”, and aspects of the planning system like: “How fast can the robot detect newly opened or closed doors and re-plan its current task schedule?”.

The visualization techniques offered by MRT-VR and RTL are well suited to investigate on these questions. For example automatically

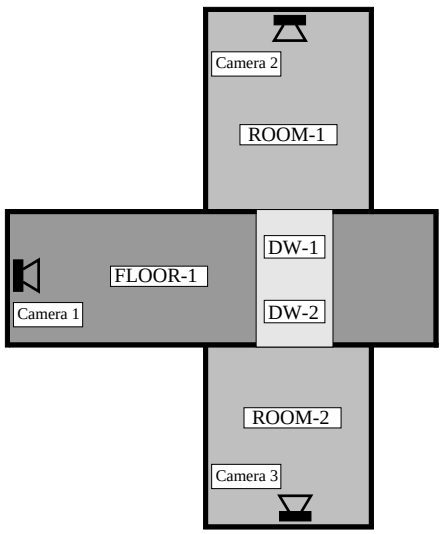

Figure 7.6: Example Environment Covered by 5 Regions of Interest

following the virtual robot enables the tele-experimenter to decide on the first question without actually having to navigate through the scene. As can be seen in row 2 of Figure 7.7 it also achieves the natural impression of walking behind the real robot, because it respects the surrounding office environment.

The automatic camera switching mechanism of RTL is especially well suited to evaluate on planning related questions. Experimenter can define regions of interest in places of the environment, which are especially important for task planning. For example the state of doors in question 2 above.

Note that camera switching can also be helpful for the observation of the navigation behavior like question 1. In our example, RTL focuses the doorway while the robot is leaving the room. Although, MRT-VR's automatic following mode is surely better suited for this particular task. MRT-VR always offers the opportunity to switch between any of the offered viewpoints: either automatic following, the viewpoint of RTL (e. g. automatic camera switching) or any one of the experimenter's selected viewpoints.

Auto-Viewpoint

The robot's cruise is determined by a web application which allows to follow a virtual tour through some of the rooms of the institute and shows the robot's capability of path-finding and collision avoidance. The whole experiment is transmitted via the Internet by capturing camera images from cameras mounted on the robot. Additionally, the robot's position is transmitted via M-Bone and MB protocol. A VRML scene of the Institute including the robot was generated and used for the simulation. Because it takes only a couple of bytes to update the robot's position via MBP and some milliseconds to draw a new picture, the image generated with MRT-VR is more accurate than a transmitted video image. In combination with dead reckoning techniques, used within the robot-control software, the simulated image could improve even more.

7.1.2 Auto Viewpoint Indicated by Simulation

Automatic camera control in virtual environments have gained some attention over the last few years in the Computer Graphics and AI communities. Drucker and Zeltzer [DD95] constrained optimization for the best viewpoint, He et al. [HCS96] declarative camera control language, constraints and rules derived from basic cinematographic rules.

Functionality which is needed during tele-experiments is limited, Important viewpoints are known at experiment design phase, the simple camera control behavior required should be easy to specify, cinematographic aspects are less important. Therefore we use a simple finite state machine based approach.

Row 1 of Figure 7.7 shows a characteristic sample situation, a tele-experimenter wants to follow the robot moving across the floor into a room. When the robot turns towards the door,

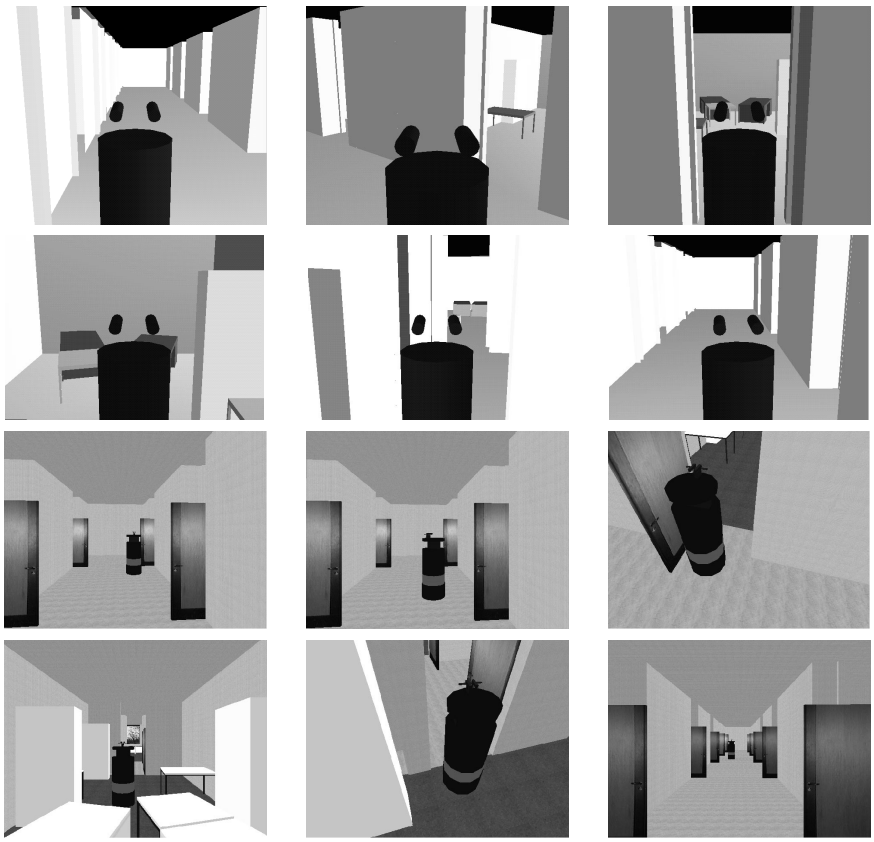

Figure 7.7: Top sequence: Automatically following the Robot; The experimenter gets the impression of walking behind the robot.Bottom sequence: RTL automatically switches viewpoints according to a selection scheme specified by the user.

the requested relative viewpoint (2 m behind the robot) will eventually move into or behind a wall. In this situation MRT-VR calculates the intersection point of the camera trajectory with the scene objects. In case of an intersection the new camera position is set at the intersection point minus an epsilon for the head size. The look vector is set to aim at the robot. The efficient implementation of these distance based navigation techniques is tremendously simplified by the ray-tracing facilities supplied by the MRT library.

RTL/MRT-VR Conclusions

The combination of RTL and MRT-VR has shown, that robot motion observation via Internet is possible. The use of dead reckoning bridges transmission gaps and leads to realistic predictions about the robot's movements. The use of Multicast protocol allows a large audience. Various camera control mechanisms allow different or automatic viewpoint selection, avoiding collisions with the 3D environment. So, the experimenter can keep focus on the experiment while other observers can follow the experiment from different viewpoints.

7.2 Sample Experiment: Particle Simulation

This chapter describes a sample application in which a particle system was connected to the MRT-VR. The particle simulator calculates the positions for 250 snowflakes, falling from top of a unit cube to its bottom, due to gravity and viscosity set up in a control file.

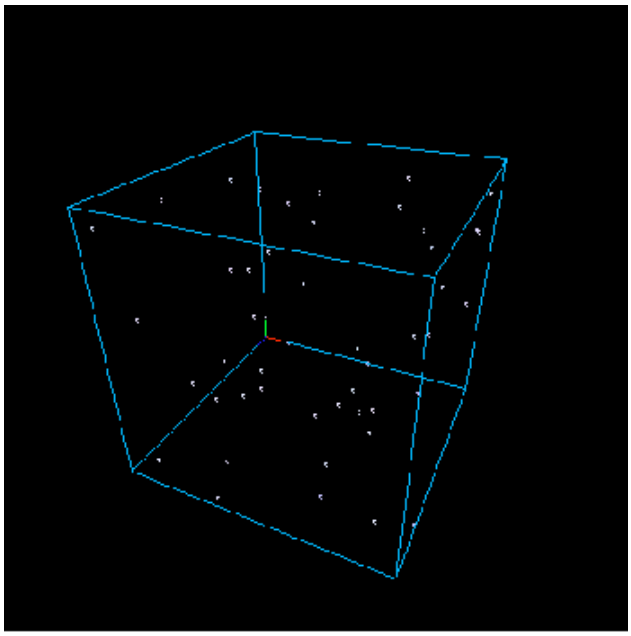

Figure 7.8: Sample Scene: Particle System (by Stefan Wolfrum) generating Snowflakes in a Unit Cube.

The approach used was of type 2, using data replication mechanism. The particle system joined the session initiated by the spectator looking at the scene. When joining the session all scene data was sent to the new participant. Figure 7.9 shows the image of the particle simulator. With the scene knowledge in memory the particle system could calculate scene dimensions and the snow area appropriate to the scene dimensions.

Snow was generated by constructing a new sub-scene, containing a reference objects pointing to one snowflake, realized as a white box.

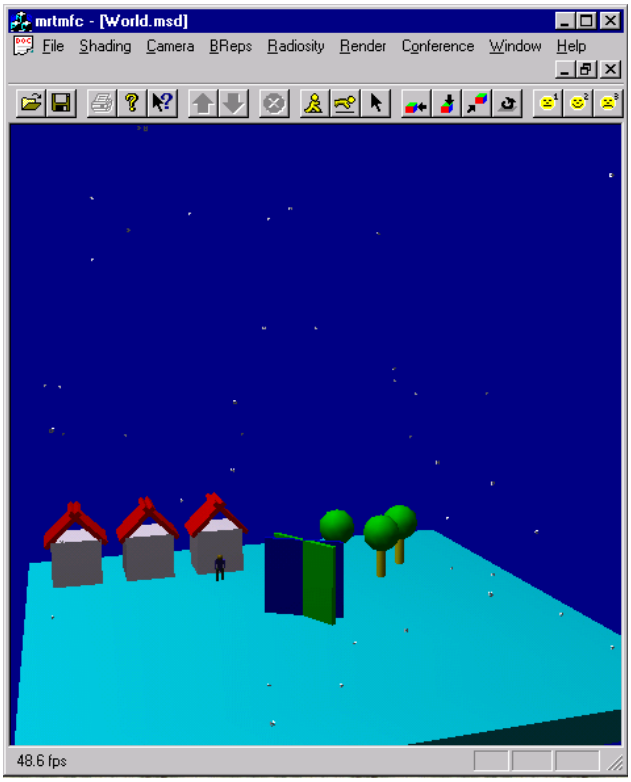

Figure 7.9: Sample Scene: MRT-VR Visualizes a Scene with Snow Particle System

This new sub-scene was inserted in the main scene. While registering the changes the new scene was propagated to the other participants.

While keeping the pointers of the reference objects in an array allowed a fast transformation of every snowflake using the reference object's transform function. To change the transformation, every object in the sub-scene should be locked. The particle system calculated the position of every particle for a specific time t. After all positions had been calculated the positions of the particles were multiplied by the generated matrix to suit in the scene, and all positions of the reference objects were updated. To propagate the changes to the other participants, the scene changes had to be 'registered' by the data management layer, calling its registerScene() method.

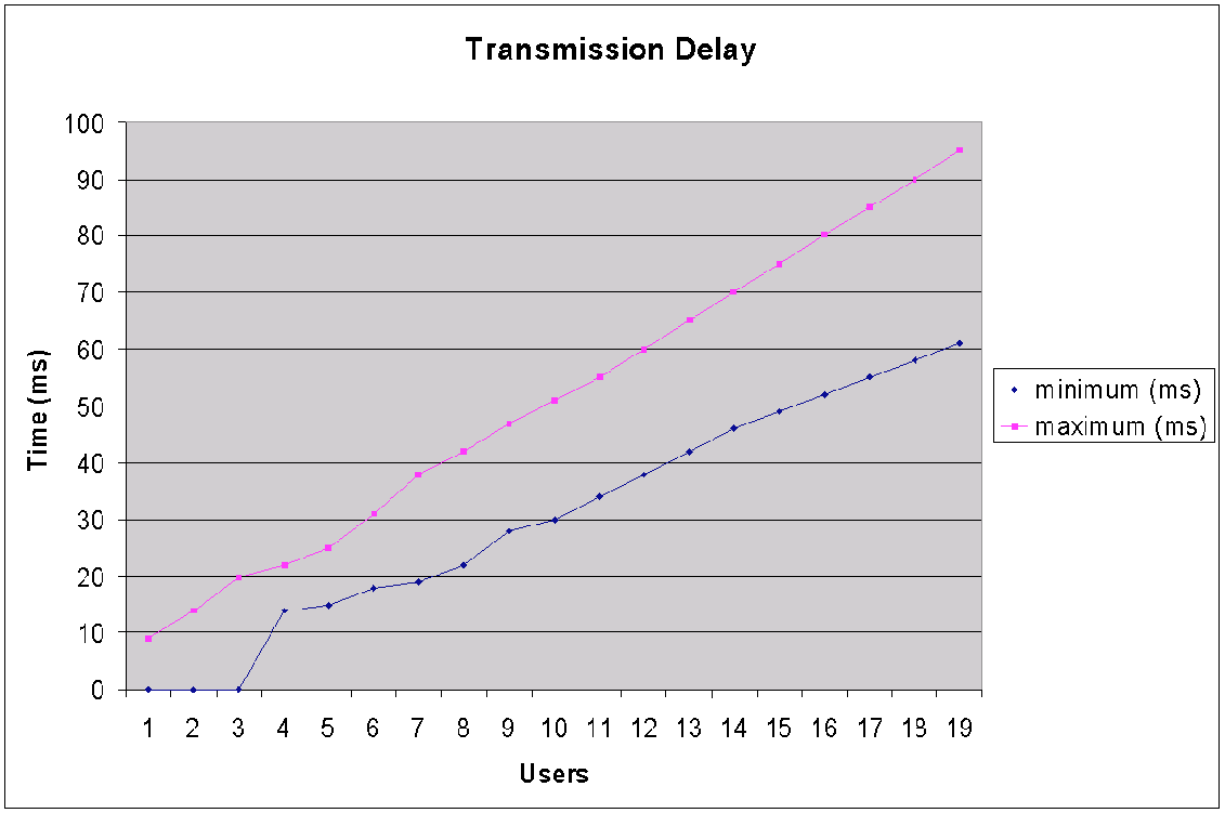

Figure 7.10: Delay of Corba Messages in Relation to the Number of Connected Users

Scene updates were transmitted 20 times a second for 250 reference objects. The generated traffic was about 48 bytes per object for its transformation matrix, leading to a data stream of about 2.5 MBit/s. If translation was sufficient, the amount of data could be reduced by using the objects translate functions, transmitting a new position vector of 12 byte, leading to a traffic of about 600kBit/s.

Further investigation has shown that for an average number of users (20) delays occuring due to the use of a centralized server scale linear with the number of the users, which is displayed in Figure 7.10. Keeping in mind that object changes were updated only ten times per second by the underlying visualization an average number of users could be provided with the scene changes without any visible delay. However, significant delays would have been visible with more than 100 participating users.

Chapter 8

Conclusion

This work presented details of a complex virtual environment called the VR-Lab. In the VR-Lab different technologies were integrated to provide tools for distributed presentation and experiments.

For the distributed presentation the software MRT-VR was developed. In combining Multicast and a distributed database graphical objects could be distributed over the internet. This improved rendering quality and, at the same time, reduced transmission bandwidth, making distributed 3D simulation possible.

A lecture room was re-built to be used as a presentation environment. Different hardware was integrated and connected to build the MMHS which served as a platform for the MRT-VR and other Multi-Media applications.

Evaluation of the VR-Lab and the MRT-VR Software with different experiments showed that the approach is flexible to provide solutions for a variety of different problems as described in the Chapter 7.

MRT-VR could be used in combination with other M-Bone tools such VIC or VAT to present 3D content to a distributed audience. For the distributed visualization, MRT-VR transports objects instead of triangles preserving the semantic information of the scene structure which provides a high compression ratio compared to triangle meshes. Preserving semantic context allows the remote systems to adjust the rendering quality to the underlying hardware without costly adjusting the quality of triangle meshes. Additionally, these images may be raytraced in an arbitrary quality. Using Corba and Multicast as an underlying data transport mechanism provides MRT-VR with a linear scaling fast transport system with additional features like access rights for scene database objects. In combination with external simulations and different data-transmission protocols MRT-VR serves as a visualization platform for experiments and can be used to present distributed experiments to a distributed audience.

The user interface and remote control system for the machines connected to the MMHS was designed, built, programmed and implemented in a lecture room at the University of Bonn. Evaluation of the MMHS system by different test persons proved a good usability.

Furthermore, the quality of 3D visualization could be improved by different approaches presented in this work. One approach was selected and evaluated in a prototype built at the University of Braunschweig and is still in use for presentations, lectures and seminars. A comparable setup was shown at the Landesausstellung in Braunschweig parallel to the Expo 2000.

In the future this project could be expanded in many directions. Due to its modular design the VR-Lab and its control software could be expanded or adapted to other lecture rooms, increasing the number of persons participating in

distributed experiments.

Several new machine type programs could be developed to perform more animations and to bring other VRML scenes to life. The MRT-VR could be expanded by supporting more object types and several special shaders. Other input devices, such as the space mouse, should be integrated in the user interface.

The most interesting development would be a combination of the proposed projection systems with the MRT-VR software used to build a multi-screen projection system (like a CAVE) from simple and cheap components. A rendering cluster of PC-based machines using MRT-VR might render different views of a VRML scene and generate the images needed for a multi-screen projection. In combination with the proposed stereoscopic projection systems the stereo images for a large audience or a CAVE system will be generated. A PC-based system used for a 3D Display was presented at the fut[o]ur Exhibition, Landesmuseum Braunschweig, from Juni 2000 to Oktober 2000.

To simplify this setup even more, further research will help to develop a single projector stereoscopic display, which might be easily mounted and adjusted, reducing the costs of the projectors by factor two.

List of Figures

1.1 The Design of the VR-LAB . . . 3
1.2 VR-LAB, the Big Picture . . . 4

2.1 Spectrum of Connectivity as seen in D.Brutzman, VRTP . . . 7
2.2 M-Bone, Map of Germany . . . 10
2.3 Design of VRTP . . . 13
2.4 Design overview of DWTP . . . 15

3.1 Design of MRT-VR . . . 20
3.2 Data Replication . . . 23
3.3 Embedding of Avatars into the Scene Structure . . . 26
3.4 MRT-VR Data Replication with the Class t_ObjectList . . . 27
3.5 MRT-VR Data-Transport-Layer with Different Transport Modules . . . 31
3.6 MBP Message Structure . . . 33
3.7 Example for a VRML File . . . 33
3.8 Code Example MSD File . . . 34
3.9 Simulation of a 3 Cylinder Motor (Model from Volkswagen AG) . . . 36
3.10 MRT-VR Application . . . 37
3.11 Integration of MRT-VR in SDR (Session Directory) . . . 39
3.12 Avatar Selection . . . 40

4.1 Parsing System . . . 42
4.2 Construction Elements for the Class t_Sphere . . . 43
4.3 Sphereflake . . . 44
4.4 Sphereflakes . . . 45
4.5 Code Example VRML File for Camera Position Definition . . . 46
4.6 Example for an Object Definition, MSD Syntax . . . 46
4.7 Example for an Object Definition for Include and Reference Calls, MSD Syntax . . . 47
4.8 Referencing an Object . . . 47

5.1 System Outline of MMHS . . . 52
5.2 MMHS Main Control Console . . . 53
5.3 MMHS Main Control Console Screen . . . 56
5.4 Speakers Control Interface . . . 57
5.5 Internet Connection Between MMHS Control PC and other Remote Machine . . . 58

5.6 Sony Camera mounted on Pan-Tilt Head . . . 62
5.7 Camera Operation Panel . . . 63
5.8 Screenshot of the Environmental Controller Slide Projector Interface . . . 64
5.9 Screenshot of the Video Projector Control Interface . . . 66

6.5 Signal Trace of Sync-Separation-Circuit . . . 68
6.1 Passive and Active Stereo Projection Systems, State of the Art . . . 69
6.2 Stereo Projection Systems, proposed passive setup . . . 69
6.3 Stereo Projection Systems, proposed active setup . . . 70
6.4 Stereo Projection Systems, proposed single projector setup . . . 70
6.6 Tracking Setup for Passive Stereo Projection . . . 71
6.7 Polarizers for Passive Stereo Projection . . . 72

7.1 The RWI B21 Robot RHINO. . . . 74
7.2 3D Visualization of Real Scenarios . . . 75
7.3 A Virtual Camera Following the Robot . . . 76
7.4 Transition Scheme for the ROF Example . . . 76
7.5 Path of the Robot During the Experiment . . . 77
7.6 Example Environment Covered by 5 Regions of Interest . . . 78
7.7 Virtual Cameras . . . 79
7.8 Sample scene: Particle System . . . 80
7.9 Sample Scene: MRT-VR Visualizes a Scene with Snow Particle System . . . 80
7.10 Delay of Corba Messages . . . 81

9.1 PIC Control PCB Layout of MMHS Environment and Camera Controller . . . 87
9.2 PIC Control PCB Layout of MMHS Audio Controller . . . 88
9.3 Screenshot of Java Remote Control Application . . . 114
9.4 Scene Grammar for the Modified VRML Parser . . . 115
9.5 Scene Grammar for the Modified MSD Parser . . . 116
9.6 Video Wiring Diagram . . . 122
9.7 Wiring Diagram for Controllers . . . 123

List of Tables

2.1 Protocols used in Virtual Environments 8

5.1 Questionnaire Determines the Usability of the System 58
5.2 Answers of Untrained Persons 59
5.3 Problems of Untrained Persons, derived from “Thinking loud” 59
5.4 Answers of Trained Persons 59
5.5 Problems of trained persons, derived from “Thinking loud” 59
5.6 Control Codes for the Environmental Controller 63
5.7 Assignment of Components to MOXA RS232 Multiplexer 66

9.1 NEC Video Projector Control 111
9.2 Phillips Commands for the VCR R 969 112
9.3 Control codes for the audio controller 112
9.4 Channel Assignment for the Audio Controller 113
9.5 Outputs of Environmental Controller Mapped to Relais 113
9.6 Channel Description of Controller 1 (Front) 117
9.7 Channel Description of Controller 2 (Back) 117
9.8 Control Codes for Camera Controller 117
9.9 Control Commands I 118
9.10 Control Commands II 119
9.11 Type Identifier of the Supported MRT Objects *Part I:* 120
9.12 Type Identifier of the Supported MRT Objects*Part II:* 121

Chapter 9

Appendix

9.1 MMHS Control Hardware

This appendix contains additional information about the hardware built for the MMHS project, because these modules are not available on the market. It shows the circuit layouts which were used to built the environmental, camera and audio controllers and the source code listings of the software programmed into the PIC microcontroller.

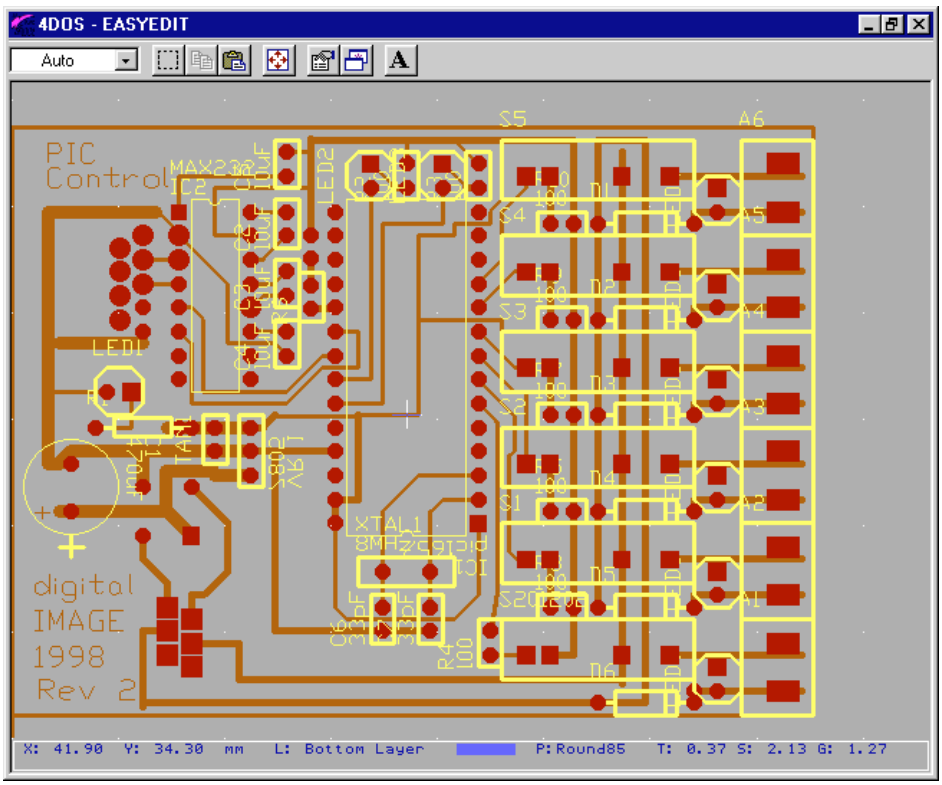

Figure 9.1: PIC Control PCB Layout of MMHS Environment and Camera Controller

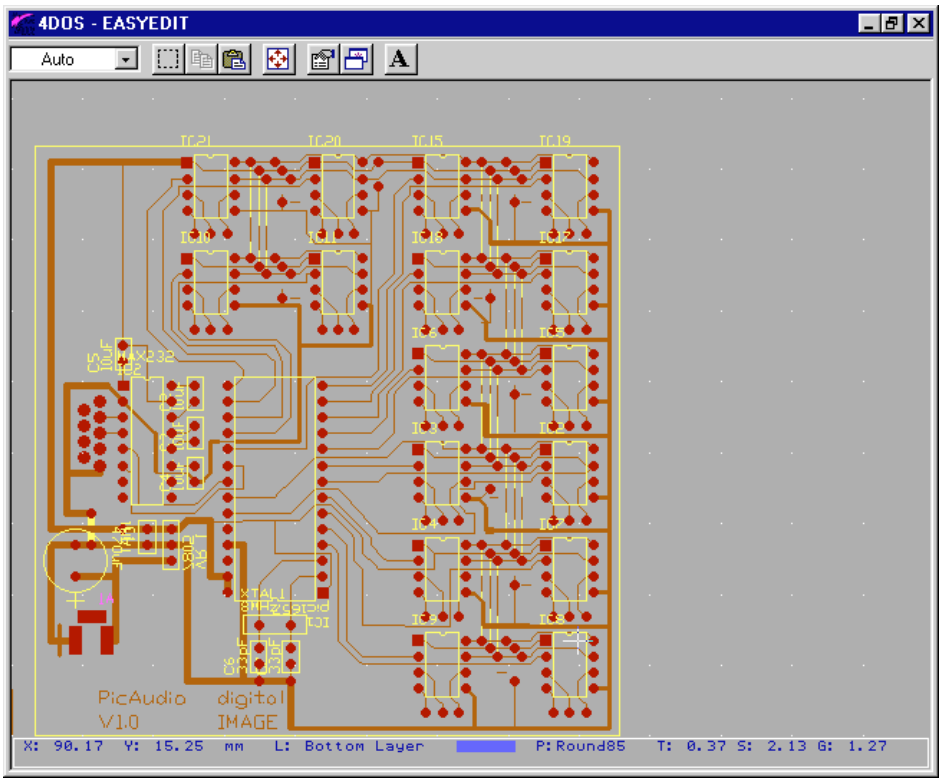

Figure 9.2: PIC Control PCB Layout of MMHS Audio Controller

9.2 MMHS Control Software

Program listing of the MMHS camera controller

```
; Receiver Software for the MMHS project (C)1997 by A.Hopp
; Receives data at 9800,8,n,1. Reads a single byte, and store it to port c
; a zero equals logic 1 and enables output
; only pins 0 to 5 of port c are used.
; infrared commands are sent at port B0, corossponding to Ralf Leonhards camera codes
;
;
;       TX -> RA3
;       RX -> RA2
;
;       Program:          SER54.ASM
;       Revision Date:
;                         12-12-97      Compatibility with MPASMWIN 1.30
;
;*****************************************************************
;
        list p=16c57
;
clockrate equ .2000000          ;define clock rate here
baudrate equ    .9600            ;define baud rate here
;
fclk  equ     clockrate/4
;*****************************************************************
```

```
;The value baudconst must be a 8 bit value only
baudconst       equ     ((fclk/baudrate)/3 - 2)
;*************************************************************
count   equ     0x10    ;used to count tx and rx data bits
txreg   equ     0x11    ;used as temp transmit register
rcreg   equ     0x11    ;received char is saved here
delay   equ     0x12    ;used to time the baud rate.

hi      equ     0x13
lo      equ     0x14
shi     equ     0x15
slo     equ     0x16
ilo     equ     0x17
ihi     equ     0x18
_rcstat equ     0x19
w45cnt  equ     0x1a
rcsave  equ     0x1b
cmdset  equ     0x1c

        include "p16c5x.inc"

\#define         ra1            PORTA,1

;
\#define _tx     PORTA,3
\#define _rx     PORTA,2
\#define _ready  PORTA,1
\#define _read   PORTA,0
\#define _irout  PORTB,0
;

        org     0
start
; TESTS
;        call    wait
;        call    Tdelay1 ; test 600 us
;        call    IR0 ; 1200 us
;        call    IR1 ; 1800Us

ende1   movlw   B'00000100'     ;RA3 = output, RA2 = input ,RA1 = output,;RA0 = output,
        tris    PORTA
        movlw   B'00001111'
        movwf   PORTA

        ; init procedure, signal device is ready
        movlw   B'00000000'    ;all out
        tris    PORTB
        movlw   B'00000000'    ;all out
        tris    PORTC
        movlw   0xff
        movwf   PORTC           ; all bits off port C

        movwf   PORTB           ; all bits off port B
        bsf     _ready          ; ready led OFF
        bsf     _read           ; clear "received" led

b2       movwf   cmdset          ; use default command set for IR transmission
mainini movlw   0xff            ; init blink counter
        movwf   shi
        movwf   rcsave          ; clear receive register
        clrf    slo
        clrf    _rcstat         ; clear received flag

main    call    receive
        btfsc   _rcstat,7
```

```
        goto    switch

        decfsz  slo, F          ; main
        goto    main
        decfsz  shi, F
        goto    main
        movlw   B'00000010'     ; toggle "ready led"
        xorwf   PORTA, 1
        bsf     _read           ; clear "received" led
        goto    mainini

switch
        btfsc   rcsave,7        ; check for IROUT, 0 in bit 7
        goto    switch1
        movfw   rcsave          ; send ir command 4 times
        call    IRxmit
        movfw   rcsave
        call    IRxmit
        movfw   rcsave
        call    IRxmit
        movfw   rcsave
        call    IRxmit

        goto    mainini

switch1 movfw   rcsave           ; get data read from serial port
        andlw   B'00001111'      ; mask lo bits for port 0-3
        movwf   PORTC            ; output data to port C

        goto    mainini

;-----------------------------------------
; TRANSMIT
;-----------------------------------------
transmit
        movwf   txreg
        bcf     _tx             ; send start bit
        movlw   baudconst
        movwf   delay
        movlw   .9
        movwf   count
txbaudwait
        decfsz  delay, F
        goto    txbaudwait
        movlw   baudconst
        movwf   delay
        decfsz  count, F
        goto    SendNextBit
        movlw   .9
        movwf   count
        bsf     _tx             ; send stop bit
        return
SendNextBit
        rrf     txreg, F
        btfss   STATUS,C
        goto    Setlo
        bsf     _tx
        goto    txbaudwait
Setlo
        bcf     _tx
        goto    txbaudwait
;-----------------------------------------
; RECEIVE
;-----------------------------------------
receive
```

```
        bcf      _rcstat,7        ; clear received flag
        btfsc    _rx              ; check i/o pin
        return
        movlw   baudconst         ; init receive time
        movwf   delay
        movlw   .9
        clrf    rcreg
        movwf   count
rxbaudwait
        decfsz  delay, F
        goto    rxbaudwait
        movlw   baudconst
        movwf   delay
        decfsz  count, F
        goto    RecvNextBit
        movfw   rcreg
        movwf   rcsave             ; save receive reg
        btfss   rcsave,7           ; check for IROUT, 0 in bit 7
        goto    sendok
        bsf     _rcstat,7          ; set received flag
        bcf     _read              ; set "received" led
        movlw   0x4f               ; send OK
        call    transmit
        call    secdelay
        call    secdelay
        movlw   0x4b
        call    transmit
        call    secdelay
        call    secdelay
        movlw   0x20
        call    transmit
        call    secdelay
        call    secdelay
        return
sendok  movlw   0x49               ; send ir
        bsf     _rcstat,7          ; set received flag
        call    transmit
        call    secdelay
        call    secdelay
        movlw   0x52
        call    transmit
        call    secdelay
        call    secdelay
        movlw   0x20
        call    transmit
        call    secdelay
        call    secdelay
        return

RecvNextBit
        bcf     STATUS,C
        btfsc   _rx
        bsf     STATUS,C
        rrf     rcreg, F
        goto    rxbaudwait
;-----------------------------------------------
; DELAY ROUTINES
;-----------------------------------------------
secdelay
        movlw   1
        movwf   hi
        clrf    lo
dly
        decfsz  lo, F ;secdelay
        goto    dly
```

```
        decfsz  hi, F
        goto    dly
        return
;-------------------------------------------
; IR ROUTINES
;-------------------------------------------
IRxmit
        movwf   txreg       ; save IR word
        call    IRBurst
        movlw   0x8
        movwf   count
IRbaudwait
        decfsz  count, F
        goto    SendNextIRBit

cmd0    call    IR1         ; "1" default command set
        call    IR0         ; "0"
        call    IR0         ; "0"
        call    IR1         ; "1"
        call    IR1         ; "1"
        call    IR0         ; "0"
        call    IR1         ; "1"
        call    IR1         ; "1"
        goto    irende
cmd1    call    IR1         ; "1" ; recorder
        call    IR1         ; "1"
        call    IR1         ; "1"
        call    IR0         ; "0"
        call    IR0         ; "0"
        goto    irende
cmd2    call    IR1         ; "1" ; send CMD Code Sony
        call    IR0         ; "0"
        call    IR0         ; "0"
        call    IR1         ; "1"
        call    IR1         ; "1"
        call    IR1         ; "1"
        call    IR0         ; "0"
        call    IR1         ; "1"
        goto    irende

irende  bsf     _irout
        call    Tdelay1

w45     movlw   0x2A;       ; wait until 45ms passed by to complete IR Code
        movwf   w45cnt
wait45  decfsz  w45cnt, F
        goto    w145
        return
w145    call    Tdelay1
        goto    wait45

SendNextIRBit
        rrf     txreg, F
        btfss   STATUS,C
        goto    IRsetlo
        call    IR1         ; "1"
        goto    IRbaudwait
IRsetlo call    IR0         ; "0"
        goto    IRbaudwait

xTdelay movlw   0x02        ; 600usec Delay
        movwf   ihi
        movlw   0x58;
        movwf   ilo
Twait   decfsz  ilo, F      ; tdelay
```

```
        goto    Twait
        decfsz  ihi, F
        goto    Twait
        return

; This sub sends an 1,0 to IR Port
; depending on call adress, it delay 600 or 1200 u
; the 1 impulse is modulated with 40kHz
IR0     movlw   0x17        ; this all should last 600us+600us delay
        goto    IRout
IRBurst movlw   0x60        ; burst should last 4x600 us+600us delay
        goto    IRout
IR1     movlw   0x2e        ; "1" this all should last 2x600us+600 delay
IRout   movwf   ihi
; -----------modulated 1 --------------------
IRwait0
        bcf     _irout      ; set IR to ON
        bcf     _read       ; set IR to ON
        nop
        nop
        nop
        bsf     _read       ; set read to off
        bsf     _irout      ; set IR to OFF
        nop
        nop
        nop
        decfsz  ihi, F
        goto    IRwait0     ; now we have modulated the "1"
                            ; we wait for 1 lo impulse time
halt1
;----------- just wait for 0 -------------------
Tdelay1 movlw   0x61         ; 600usec Delay
        movwf   ilo
IRwait  decfsz  ilo, F
        goto    IRwait
;        decfsz  ihi, F
;        goto    IRwait
halt3   return

        end
```

Program Listing of the audio control module

```
; Receiver Software for the MMHS project (C)1997 by A.Hopp
; Receives data at 9800,8,n,1. Reads a single byte, to control the audio outputs
; 14 D/A controllers are used. Every CS of the D/A converters is connected with PortB and C
; clock and direction (up and down) is common for all D/A converters and connected to Port A0 and A1
; Command set:

; lo nibble : chip select, (0-16) o-8 portB 8-16 port C
; hi nibble : 0  - reset, all channels mute (lo nibble without function=reset)
;             1  -        increase 1, lo nibble selects channel
;             2 -         decrease 1, lo nibble selects channel
;             4 -         increase 5, lo nibble selects channel
;             8 -         decrease 5, lo nibble selects channel
; CS aktiv lo, init mit hi
; inc is aktiv lo, also init hi
;       TX -> RA3
;       RX -> RA2
;
;       Program:          SER54.ASM
;       Revision Date:
;                         12-12-97      Compatibility with MPASMWIN 1.30
;
;*****************************************************************
;
        list p=16c57
;
clockrate equ .2000000          ;define clock rate here
baudrate equ    .9600           ;define baud rate here
;
fclk  equ     clockrate/4
;*********************************************************
;The value baudconst must be a 8 bit value only
baudconst       equ     ((fclk/baudrate)/3 - 2)
;*********************************************************
count   equ     0x10    ;used to count tx and rx data bits
txreg   equ     0x11    ;used as temp transmit register
rcreg   equ     0x11    ;received char is saved here
delay   equ     0x12    ;used to time the baud rate.

hi      equ     0x13
lo      equ     0x14
shi     equ     0x15
slo     equ     0x16
ilo     equ     0x17
ihi     equ     0x18
s_port  equ     0x19
s_shift equ     0x1A

        include "p16c5x.inc"

\#define _tx     PORTA,3
\#define _rx     PORTA,2
\#define _INC    PORTA,1
\#define _UP     PORTA,0

        org     0
start

ende1   movlw   B'00000100'     ;RA3 = output, RA2 = input ,RA1 = output,;RA0 = output,
        tris    PORTA
        bsf     _INC            ;takt auf hi
        bsf     _UP             ;richtung ist up
```

```
;init procedure
        movlw   B'00000000'      ;all out
        tris    PORTB
        tris    PORTC
        call    selectnone       ; all CS to hi, so that no chip is doing anything !
        call    muteAll
;debugging
;        movlw   B'00000000'     ; all mute
;        movlw   B'00010001'     ; dec channel 1 by one, Port B, chip 1
;        movlw   B'00100001'     ; inc channel 1 by one, Port B, chip 1
;        movlw   B'01000001'     ; dec channel 1 by five, Port B, chip 1
;        movlw   B'10000001'     ; inc channel 1 by five, Port B, chip 1
;        movlw   B'00011001'     ; dec channel 9 by one, Port C, chip 1
;        movlw   B'00010111'     ; dec channel 8 by one, Port B, chip 7
;        movlw   B'00010000'     ; dec channel 8 by one, Port B, chip 0
;        movwf   rcreg
;        call    debug

main
        call    receive
        goto    mainini

;----------------------------------------
; TRANSMIT
;----------------------------------------
transmit
        movwf   txreg
        bcf     _tx             ; send start bit
        movlw   baudconst
        movwf   delay
        movlw   .9
        movwf   count
txbaudwait
        decfsz  delay, F
        goto    txbaudwait
        movlw   baudconst
        movwf   delay
        decfsz  count, F
        goto    SendNextBit
        movlw   .9
        movwf   count
        bsf     _tx             ; send stop bit
        return
SendNextBit
        rrf     txreg, F
        btfss   STATUS,C
        goto    Setlo
        bsf     _tx
        goto    txbaudwait
Setlo
        bcf     _tx
        goto    txbaudwait
;----------------------------------------
; RECEIVE
;----------------------------------------
receive
        btfsc    _rx             ; check i/o pin
        return
        movlw   baudconst        ; init receive time
        movwf   delay
rxbaudwait
        decfsz  delay, F
        goto    rxbaudwait
        movlw   baudconst
        movwf   delay
```

```
        decfsz  count, F
        goto    RecvNextBit
        movlw   .9              ; restore receive status
        movwf   count
                                ; data is now in rcreg
                                ; select desired channel...
debug   movfw   rcreg           ; get data read from serial port
        andlw   0x07            ; select 0-8 aoutput bit
        movwf   s_port
        incf    s_port,1

        BSF     STATUS,C        ; set carry for first audio chip to roll in
        clrf    s_shift
rjmp    rlf     s_shift,1       ; shift register
        decfsz  s_port,1        ; while portcount > 0
        goto    rjmp
        comf    s_shift,1       ; complement, because of inverse logic
        movfw   rcreg           ; load received word
                                ; now select correct Bank (portB or Port C)
        btfsc   rcreg,3         ; check Port B or C
        goto    s_etC
        movfw   s_shift         ; set selected portB
        movwf   PORTB
        goto    cont
s_etC    movfw   s_shift         ; set selected portC

        movwf   PORTC
                                ; now select right command
cont
        btfsc   rcreg,7         ; plusFive
        goto    plusFive
        btfsc   rcreg,6         ; minusFive
        goto    minusFive
        btfsc   rcreg,5         ; plusOne
        goto    plusOne
        btfsc   rcreg,4         ; minusOne
        goto    minusOne
        goto    muteAll         ; noBit set, so mute!
        return

RecvNextBit
        bcf     STATUS,C
        btfsc   _rx
        bsf     STATUS,C
        rrf     rcreg, F
        goto    rxbaudwait
;-------------------------------------------
; DELAY ROUTINES
;-------------------------------------------
secdelay
        movlw   1
        movwf   hi
        clrf    lo
dly
        decfsz  lo, F ;secdelay
        goto    dly
        decfsz  hi, F
        goto    dly
        return

plusOne
        bcf     _UP ; go up
        bcf     _INC
        bsf     _INC ; one step
        goto    selectnone
```

```
plusFive
        movlw   5
        movwf   lo
        bcf     _UP ; go up
dly4    bcf     _INC
        bsf     _INC ; one step
        decfsz  lo, F ;secdelay
        goto    dly4
        goto    selectnone

minusOne
        bsf     _UP ; go down
        bcf     _INC
        bsf     _INC ; one step
        goto    selectnone

minusFive
        movlw   5
        movwf   lo
        bsf     _UP ; go down
dly2    bcf     _INC
        bsf     _INC ; one step
        decfsz  lo, F ;secdelay
        goto    dly2
        goto    selectnone

muteAll clrf    PORTB ; select all CHIPS
        clrf    PORTC
        movlw   .101
        movwf   lo
        bsf     _UP ; go down
dly1    bcf     _INC
        bsf     _INC ; one step
        decfsz  lo, F ;secdelay
        goto    dly1
        goto    selectnone

; all CS to hi, so that no chip is doing anything !
selectnone
        movlw   0xff
        movwf   PORTB
        movwf   PORTC
        return
        end
```

Program Listing of the Environment Control Module

```
;
; Das ist die nicht interrupt gesteurte version der uni umweltkontrolle
; ein byte wird empfangen 9800,8,n,1 und die entsrpechenden bits
; am ausgang (PORTC) gesetzt. Dabei entspricht ein 0 bit einer logisch 1
; Ausg"nge 6 Bit 0-5, hi bits 6 und 7 nicht benutzt.
;
;       TX -> RA3
;       RX -> RA2
;
;       Program:          SER54.ASM
;       Revision Date:
;                         12-12-97      Compatibility with MPASMWIN 1.30
;
;*****************************************************************
;
        list p=16c57
;
clockrate equ .2000000         ;define clock rate here
baudrate equ    .9600           ;define baud rate here
;
fclk  equ     clockrate/4
;*********************************************************
;The value baudconst must be a 8 bit value only
baudconst       equ     ((fclk/baudrate)/3 - 2)
;*********************************************************
count   equ     0x10    ;used to count tx and rx data bits
txreg   equ     0x11    ;used as temp transmit register
rcreg   equ     0x11    ;received char is saved here
delay   equ     0x12    ;used to time the baud rate.

hi      equ     0x13
lo      equ     0x14
shi     equ     0x15
slo     equ     0x16
ilo     equ     0x17
ihi     equ     0x18
_rcstat equ     0x19
w45cnt  equ     0x1a
rcsave  equ     0x1b
cmdset  equ     0x1c

        include "p16c5x.inc"

;
\#define _tx     PORTA,3
\#define _rx     PORTA,2
\#define _ready  PORTA,1
\#define _read   PORTA,0
\#define _irout  PORTB,0

;

        org     0
start
; TESTS
;        call    wait
;        call    Tdelay1 ; test 600 us
;        call    IR0 ; 1200 us
;        call    IR1 ; 1800Us

ende1   movlw   B'00000100'     ;RA3 = output, RA2 = input ,RA1 = output,;RA0 = output,
        tris    PORTA
```

```
        movlw   B'00001111'
        movwf   PORTA

        ; init procedure, signal device is ready
        movlw   B'00000000'    ;all out
        tris    PORTB
        movlw   B'00000000'    ;all out
        tris    PORTC
        movlw   0xff
        movwf   PORTC           ; all bits off port C

        movwf   PORTB           ; all bits off port B
        bsf     _ready          ; ready led OFF
        bsf     _read           ; clear "received" led

b2       movwf   cmdset           ; use default command set for IR transmission
mainini movlw   0xff            ; init blink counter
        movwf   shi
        movwf   rcsave          ; clear receive register
        clrf    slo
        clrf    _rcstat         ; clear received flag

main    call    receive
        btfsc   _rcstat,7
        goto    switch

        decfsz  slo, F          ; main
        goto    main
        decfsz  shi, F
        goto    main
        movlw   B'00000010'     ; toggle "ready led"
        xorwf   PORTA, 1
        bsf     _read           ; clear "received" led
        goto    mainini

switch
        btfsc   rcsave,7        ; check for IROUT, 0 in bit 7
        goto    switch1
        movfw   rcsave          ; send ir command 4 times
        call    IRxmit
        movfw   rcsave
        call    IRxmit
        movfw   rcsave
        call    IRxmit
        movfw   rcsave
        call    IRxmit

        goto    mainini

switch1 movfw   rcsave           ; get data read from serial port
        andlw   B'00001111'      ; mask lo bits for port 0-3
        movwf   PORTC            ; output data to port C
;       bcf     _read            ; set "received" led

        goto    mainini

;-----------------------------------------
; TRANSMIT
;-----------------------------------------
transmit
        movwf   txreg
        bcf     _tx             ; send start bit
        movlw   baudconst
        movwf   delay
        movlw   .9
```

```
        movwf   count
txbaudwait
        decfsz  delay, F
        goto    txbaudwait
        movlw   baudconst
        movwf   delay
        decfsz  count, F
        goto    SendNextBit
        movlw   .9
        movwf   count
        bsf     _tx             ; send stop bit
        return
SendNextBit
        rrf     txreg, F
        btfss   STATUS,C
        goto    Setlo
        bsf     _tx
        goto    txbaudwait
Setlo
        bcf     _tx
        goto    txbaudwait
;-----------------------------------------
; RECEIVE
;-----------------------------------------
receive
        bcf      _rcstat,7        ; clear received flag
        btfsc    _rx              ; check i/o pin
        return
        movlw   baudconst         ; init receive time
        movwf   delay
        movlw   .9
        clrf    rcreg
        movwf   count
rxbaudwait
        decfsz  delay, F
        goto    rxbaudwait
        movlw   baudconst
        movwf   delay
        decfsz  count, F
        goto    RecvNextBit
        movfw   rcreg
        movwf   rcsave             ; save receive reg
        btfss   rcsave,7           ; check for IROUT, 0 in bit 7
        goto    sendok
        bsf     _rcstat,7          ; set received flag
        bcf     _read              ; set "received" led
        movlw   0x4f               ; send OK
        call    transmit
        call    secdelay
        call    secdelay
        movlw   0x4b
        call    transmit
        call    secdelay
        call    secdelay
        movlw   0x20
        call    transmit
        call    secdelay
        call    secdelay
        return
sendok  movlw   0x49               ; send ir
        bsf     _rcstat,7          ; set received flag
        call    transmit
        call    secdelay
        call    secdelay
        movlw   0x52
```

```
        call    transmit
        call    secdelay
        call    secdelay
        movlw   0x20
        call    transmit
        call    secdelay
        call    secdelay
        return

RecvNextBit
        bcf     STATUS,C
        btfsc   _rx
        bsf     STATUS,C
        rrf     rcreg, F
        goto    rxbaudwait
;-----------------------------------------------
; DELAY ROUTINES
;-----------------------------------------------
secdelay
        movlw   1
        movwf   hi
        clrf    lo
dly
        decfsz  lo, F ;secdelay
        goto    dly
        decfsz  hi, F
        goto    dly
        return


;-----------------------------------------------
; IR ROUTINES
;-----------------------------------------------
IRxmit
        movwf   txreg       ; save IR word
        call    IRBurst
        movlw   0x8
        movwf   count
IRbaudwait
        decfsz  count, F
        goto    SendNextIRBit
cmd0    call    IR1         ; "1" default command set
        call    IR0         ; "0"
        call    IR0         ; "0"
        call    IR1         ; "1"
        call    IR1         ; "1"
        call    IR0         ; "0"
        call    IR1         ; "1"
        call    IR1         ; "1"
        goto    irende
cmd1    call    IR1         ; "1" ; recorder
        call    IR1         ; "1"
        call    IR1         ; "1"
        call    IR0         ; "0"
        call    IR0         ; "0"
        goto    irende
cmd2    call    IR1         ; "1" ; send CMD Code Sony
        call    IR0         ; "0"
        call    IR0         ; "0"
        call    IR1         ; "1"
        call    IR1         ; "1"
        call    IR1         ; "1"
        call    IR0         ; "0"
        call    IR1         ; "1"
        goto    irende
```

```
irende  bsf     _irout
        call    Tdelay1

w45     movlw   0x2A;       ; wait until 45ms passed by to complete IR Code
        movwf   w45cnt
wait45  decfsz  w45cnt, F
        goto    w145
        return
w145    call    Tdelay1
        goto    wait45

SendNextIRBit
        rrf     txreg, F
        btfss   STATUS,C
        goto    IRsetlo
        call    IR1         ; "1"
        goto    IRbaudwait
IRsetlo call    IR0         ; "0"
        goto    IRbaudwait

xTdelay movlw   0x02        ; 600usec Delay
        movwf   ihi
        movlw   0x58;
        movwf   ilo
Twait   decfsz  ilo, F      ; tdelay
        goto    Twait
        decfsz  ihi, F
        goto    Twait
        return

; This sub sends an 1,0 to IR Port
; depending on call adress, it delay 600 or 1200 u
; the 1 impulse is modulated with 40kHz
IR0     movlw   0x17        ; this all should last 600us+600us delay
        goto    IRout
IRBurst movlw   0x60        ; burst should last 4x600 us+600us delay
        goto    IRout
IR1     movlw   0x2e        ; "1" this all should last 2x600us+600 delay
IRout   movwf   ihi
; -----------modulated 1 --------------------
IRwait0
        bcf     _irout      ; set IR to ON
        bcf     _read       ; set IR to ON
        nop
        nop
        nop
        bsf     _read       ; set read to off
        bsf     _irout      ; set IR to OFF
        nop
        nop
        nop
        decfsz  ihi, F
        goto    IRwait0     ; now we have modulated the "1"
                            ; we wait for 1 lo impulse time
halt1
;----------- just wait for 0 -------------------
Tdelay1 movlw   0x61         ; 600usec Delay
        movwf   ilo
IRwait  decfsz  ilo, F
        goto    IRwait
halt3   return
        end
```

Java applet for the remote control

```
//*************************************************************************
// JRemote.java:        Applet
//
//*************************************************************************
import java.applet.*;
import java.awt.*;
import java.net.*;
import java.io.*;

//=========================================================================
// Applet JRemote
//
//=========================================================================
public class JRemote extends Applet implements Runnable
{
        // THREAD-
        //              m_JRemote       Thread-Object
        //-------------------------------------------------------------
        private Thread   m_JRemote = null;

        // animation support:
        //              m_Graphics              save graphics context
        //              m_Images[]              image array for animation
        //              m_nCurrImage            index to next image
        //              m_ImgWidth              image width
        //              m_ImgHeight             image height
        //              m_fAllLoaded            indicates if all files were laoded
        //              NUM_IMAGES              number of images for animation
        //-------------------------------------------------------------
        private Graphics m_Graphics;
        private boolean  m_fAllLoaded = false;

        // BtnPic Array
        private Image    m_BtnPics[];
        private boolean  m_BtnStatus;

        // TCP/IP Client - Socket
        Socket           TCPClient;
        InetAddress      address;
        InputStream      inStr;
        OutputStream     outStr;
        byte[]           message = new byte[256];
        byte[]           inmessage = new byte[256];


        //-------------------------------------------------------------
        private String m_IP = "160.160.160.5";
        private String m_Port = "5000";
        private boolean m_AutoStatus = true;
        private String m_BtnTyp = "btClick";
        private String m_CommandOn = "";
        private String m_CommandOff = "";
        private String m_Caption = "";
        private String m_BtnPic = "bpLight";

        //-------------------------------------------------------------
        private final String PARAM_IP = "IP";
        private final String PARAM_Port = "Port";
        private final String PARAM_AutoStatus = "AutoStatus";
        private final String PARAM_BtnTyp = "BtnTyp";
        private final String PARAM_CommandOn = "CommandOn";
        private final String PARAM_CommandOff = "CommandOff";
```

```
    private final String PARAM_Caption = "Caption";
    private final String PARAM_BtnPic = "BtnPic";

    // JRemote Class Constructor
    //---------------------------------------------------------------
    public JRemote()
    {

    }

    // APPLET-INFO

    //---------------------------------------------------------------
    public String getAppletInfo()
    {
            return "Name: JRemote\r\n" +
                   "Autor/Autorin: Ralf Leonhard\r\n" +
                   "Erstellt mit Microsoft Visual J++ Version 1.1\r\n" +
                   "\r\n" +
                   "Java Remote for MMHS\r\n" +
                   "";
    }


// JRemote Parameter-Information:
//  { "Name", "Type", "Description" },
//---------------------------------------------------------------
    public String[][] getParameterInfo()
    {
            String[][] info =
            {
                    { PARAM_IP, "String", "IP Adress of Server" },
                    { PARAM_Port, "String", "PortAdress of Server" },
                    { PARAM_AutoStatus, "boolean", "Status change automatically'' },
                    { PARAM_BtnTyp, "String", "Key or Button" },
                    { PARAM_CommandOn, "String", "Command  for turn on" },
                    { PARAM_CommandOff, "String", "Command for turn off" },
                    { PARAM_Caption, "String", "Caption" },
                    { PARAM_BtnPic, "String", "image" },
            };
            return info;
    }


//---------------------------------------------------------------
    public void init()
    {

            //---------------------------------------------------------------
            String param;

            // IP: IP Adress of the Server
            //---------------------------------------------------------------
            param = getParameter(PARAM_IP);
            if (param != null)
                    m_IP = param;

            // Port: PortAdress of the Server
            //---------------------------------------------------------------
            param = getParameter(PARAM_Port);
            if (param != null)
                    m_Port = param;

            // AutoStatus: change status automatically
            //---------------------------------------------------------------
```

```
            param = getParameter(PARAM_AutoStatus);
            if (param != null)
                    m_AutoStatus = Boolean.valueOf(param).booleanValue();

            // BtnTyp: key or button
            //---------------------------------------------------------------
            param = getParameter(PARAM_BtnTyp);
            if (param != null)
                    m_BtnTyp = param;

            // CommandOn: command string  for pushon
            //---------------------------------------------------------------
            param = getParameter(PARAM_CommandOn);
            if (param != null)
                    m_CommandOn = param;

            // CommandOff: command string for realease
            //---------------------------------------------------------------
            param = getParameter(PARAM_CommandOff);
            if (param != null)
                    m_CommandOff = param;

            // Caption:
            //---------------------------------------------------------------
            param = getParameter(PARAM_Caption);
            if (param != null)
                    m_Caption = param;

            // BtnPic: image
            //---------------------------------------------------------------
            param = getParameter(PARAM_BtnPic);
            if (param != null)
                    m_BtnPic = param;



    }


    public void destroy()
    {

    }

// animation support :
//          paint next image, when all images are loaded
//---------------------------------------------------------------
    private void displayImage(Graphics g)
    {
            if (!m_fAllLoaded)
                    return;

            // Paint picture on top left
            //---------------------------------------------------------------
            if (m_BtnStatus)
            {
                    g.drawImage(m_BtnPics[0], 0, 0, null);
            } else {
                    g.drawImage(m_BtnPics[1], 0, 0, null);
            }

    }

    // JRemote paint routine
    //---------------------------------------------------------------
```

```
public void paint(Graphics g)
{
        // animation support
        // shows message while all images are beeing loaded
        //-----------------------------------------------------------------
        if (m_fAllLoaded)
        {
                Rectangle r = g.getClipRect();

                g.clearRect(r.x, r.y, r.width, r.height);
                displayImage(g);
        }
        else
                g.drawString("Loading...", 2, 20);


}

// start is called when applet is started for the first time when shown on a html page
//-----------------------------------------------------------------
public void start()
{
        if (m_JRemote == null)
        {
                m_JRemote = new Thread(this);
                m_JRemote.start();
        }

}

// stop is called when applet dissapeares from screen.
//-----------------------------------------------------------------
public void stop()
{
        if (m_JRemote != null)
        {
                m_JRemote.stop();
                m_JRemote = null;
        }
}

//-----------------------------------------------------------------
public void run()
{
        boolean oldBtnStatus = true;
        m_BtnStatus = false;


        // m_fAllLoaded == TRUE.
        //-----------------------------------------------------------------
if (!m_fAllLoaded)
        {
        repaint();
        m_Graphics = getGraphics();
                m_BtnPics = new Image[2];

        // load the pictures
        //-----------------------------------------------------------
        MediaTracker tracker = new MediaTracker(this);
        String strImage;


        //-----------------------------------------------------------

                strImage = "images/"+m_BtnPic+"_on.gif";
```

```
                m_BtnPics[0] = getImage(getDocumentBase(), strImage);
        tracker.addImage(m_BtnPics[0], 0);
                strImage = "images/"+m_BtnPic+"_off.gif";
                m_BtnPics[1] = getImage(getDocumentBase(), strImage);
        tracker.addImage(m_BtnPics[1], 0);

                ////////////////////////////////////////////////////////////////
                // Connect to Remote server
                // -------------------------------------------------------
                try {
                        address = InetAddress.getByName(m_IP);
                        TCPClient = new Socket(address, 5000);
                        inStr = new DataInputStream(TCPClient.getInputStream());
                        outStr = new DataOutputStream(TCPClient.getOutputStream());

                        outStr.write(message);
                }
                catch (SocketException e) {
                        m_Graphics.drawString("Socket Exception", 2, 60);
                }
                catch (UnknownHostException e) {
                        m_Graphics.drawString("Unknown Host", 2, 60);
                }
                catch (IOException e) {
                        m_Graphics.drawString("IO init", 2, 60);
                };

                // Wait until pics loaded
                //-------------------------------------------------------
                try
                {
                        tracker.waitForAll();
                        m_fAllLoaded = !tracker.isErrorAny();
                }
                catch (InterruptedException e)
                {
                 }

                if (!m_fAllLoaded)
                {
                    stop();
                    m_Graphics.drawString("Error while loading the images!", 10, 40);
                    return;
                }
}
        repaint();

        String tmpStr;
        while (true)
        {
                try
                {
                        // paint pictures
                        //-------------------------------------------------------
                        if (m_BtnStatus != oldBtnStatus) {
                                displayImage(m_Graphics);
                                oldBtnStatus = m_BtnStatus;
                        }

                        try
                        {
                           inStr.read(inmessage);
                           if (m_AutoStatus) {
                                   tmpStr = new String(inmessage, 256);
                                   if (tmpStr.indexOf(m_CommandOn) != -1) {
```

```
                                        m_BtnStatus = true;
                                    } else if (tmpStr.indexOf(m_CommandOff) != -1) {
                                        m_BtnStatus = false;
                                    }
                                }
                        }
                        catch (IOException e)
                        {
                                m_Graphics.drawString("IO inStr", 2, 60);
                        };
                        Thread.sleep(50);
                }
                catch (InterruptedException e)
                {
                        stop();
                }
        }
}

// MOUSE Support
// mouseDown() is called on mousedown, when mouse is over applet
//-----------------------------------------------------------------
public boolean mouseDown(Event evt, int x, int y) {
        if (m_BtnTyp.equals("btUpDown")) {
                try {
                        if (m_CommandOn != "") {
                           m_CommandOn.getBytes(0, m_CommandOn.length(), message, 0);
                           message[m_CommandOn.length()] = 0;
                           outStr.write(message);
                        }
                        m_BtnStatus = true;
                        displayImage(m_Graphics);
                }
                catch (SocketException e) {}
                catch (UnknownHostException e) {}
                catch (IOException e) {};
        }
        return true;
}


// MOUSE CONTROL
// Mouseup called on mouse release when mouse is over applet
//-----------------------------------------------------------------
public boolean mouseUp(Event evt, int x, int y) {
        if (m_BtnTyp.equals("btUpDown")) {
                try {
                        if (!m_CommandOff.equals("")) {
                           m_CommandOff.getBytes(0,
 m_CommandOff.length(), message, 0);
                           message[m_CommandOff.length()] = 0;
                           outStr.write(message);
                        }
                        m_BtnStatus = false;
                        displayImage(m_Graphics);
                }
                catch (SocketException e) {}
                catch (UnknownHostException e) {}
                catch (IOException e) {};
        } else if (m_BtnTyp.equals("btClick")) {
                if (m_BtnStatus) {
                   try {
                           if (!m_CommandOff.equals("")) {
                              m_CommandOff.getBytes(0,
m_CommandOff.length(), message, 0);
```

```
                                    message[m_CommandOff.length()] = 0;
                                    outStr.write(message);
                                }
                                m_BtnStatus = false;
                                displayImage(m_Graphics);
                            }
                            catch (SocketException e) {}
                            catch (UnknownHostException e) {}
                            catch (IOException e) {};
                        } else {
                            try {
                                if (!m_CommandOn.equals("")) {
                                    m_CommandOn.getBytes(0, m_CommandOn.length(), message, 0);
                                    message[m_CommandOn.length()] = 0;
                                    outStr.write(message);
                                }
                                m_BtnStatus = true;
                                displayImage(m_Graphics);
                            }
                            catch (SocketException e) {}
                            catch (UnknownHostException e) {}
                            catch (IOException e) {};
                        }
                    }

                    return true;
                }
            }
```

Sample HTML document to activate the control applets

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
   <TITLE>JRemote
   </TITLE>
</HEAD>

<BODY BGCOLOR="#202020" FGCOLOR="#f0f0f0">
<HR>

<TABLE BORDER COLS=2 WIDTH="100%" >
<TR>VCR1
<TD><P><img src="images/STOP_ON.gif" >
#<FONT COLOR="#FFFFFF"></FONT>Recorder 1 Play
    <APPLET CODE="JRemote.class" NAME="JRemote" WIDTH=39 HEIGHT=39 ALIGN="ABSMIDDLE">
      <PARAM  NAME="IP" VALUE="134.169.9.114">
      <PARAM  NAME="Port" VALUE="5001">
      <PARAM NAME="AutoStatus" VALUE="true">
      <PARAM NAME="BtnTyp" VALUE="btClick">
      <PARAM NAME="CommandOn" VALUE="RECORDER1 PLAY ON">
      <PARAM NAME="CommandOff" VALUE="RECORDER1 PLAY OFF">
      <PARAM NAME="Caption" VALUE="">
      <PARAM NAME="BtnPic" VALUE="Play">
    </APPLET>
</TD>
<TD><P><FONT COLOR="#FFFFFF"></FONT>
    <APPLET CODE="JRemote.class" NAME="JRemote" WIDTH=39 HEIGHT=39 ALIGN="ABSMIDDLE">
      <PARAM NAME="IP" VALUE="134.169.9.114">
      <PARAM NAME="Port" VALUE="5001">
      <PARAM NAME="AutoStatus" VALUE="false">
      <PARAM NAME="BtnTyp" VALUE="btClick">
      <PARAM NAME="CommandOn" VALUE="RECORDER1 STOP ON">
      <PARAM NAME="CommandOff" VALUE="RECORDER1 STOP OFF">
      <PARAM NAME="Caption" VALUE="">
      <PARAM NAME="BtnPic" VALUE="Stop">
    </APPLET>
</TD>
</TR>
</TABLE>

<FONT COLOR="#FFFFFF"></FONT>
<P> DEMO PAGE
</BODY>
</HTML>
```

9.3 Hardware Control Codes for Third Party Hardware

This section contains the control code sequences for machines available on the market and used for the MMHS project. The codes and theis functions are listed in the following tables.

The NEC 1000 projectors are conrolled using the following codes:

COMMAND	BYTE VALUE
SOURCE Select	
PROJ VIDEO	3
PROJ RGB1	4
PROJ RGB2	5
PROJ SVIDEO	6
Control	
ZOOM IN	9
ZOOM OUT	0xA
FOCUS IN	0xB
FOCUS OUT	0xC
POWER OFF	0x14
MUTE	0x47
OSD OFF	0x11
RESET	0x43
DIGIZOOM IN	0x89
DIGIZOOM OUT	0x8A
MOVE UP	0xDE
MOVE DOWN	0xDF
MOVE RIGHT	0xDC
MOVE LEFT	0xDD
MOVE UPRIGHT	0x21
MOVE UPLEFT	0x22
MOVE DOWNRIGHT	0x23
MOVE DOWNLEFT	0x24
BRIGHT	0x60 value (0..64)
CONTRAST	0x62 value (0..64)
COLOR	0x64 value (0..64)
FOCUS	0x68 value (0..64)

Table 9.1: NEC Video Projector Control

All projector commands have to be terminated by a CR (0x13) byte.

Each command has to be terminated with a 0x13 (CR). The SA command returns the current mode of the VCR, do not use this too often VCR is blocked by this command.

ASCII String	Command
PL	PLAY
PS	PLAY STILL
ST	STOP
RC	RECORD
FF8	SEARCH FORWARD
RW8	SEARCH BACKWARD
FF	FAST FORWARD
RW	FAST REWIND
SA	GET STATUS

Table 9.2: Phillips Commands for the VCR R 969

9.4 Hardware Control Codes for Custom Hardware

This section contains the Control Code sequences for Machines which built for the MMHS project. The codes and theis functions are listed in the following tables.

Control Bits	Command
0x0000 0000	Mute All Channels
0x0001 nnnn	decrease Channel n by one
0x0010 nnnn	increase Channel n by one
0x0100 nnnn	decrease Channel n by five
0x1000 nnnn	increase Channel n by five

Table 9.3: Control codes for the audio controller

Controller	Audio Mixer Channel
0	6 A-Bus (Speakers)
1	NC
2	NC
3	Reverb
4	Master Volume
5	6 B-Bus (External)
6	5 A-Bus (Speakers)
7	5 B-Bus (External)
8	4 A-Bus (Speakers)
9	4 B-Bus (External)
10	3 A-Bus (Speakers)
11	3 B-Bus (External)
12	2 A-Bus (Speakers)
13	2 B-Bus (External)
14	1 A-Bus (Speakers)
15	1 B-Bus (External)

Table 9.4: Channel Assignment for the Audio Controller

Output	Signal
2	Relay 0 +
3	Relay 0 -
4	Relay 1 +
5	Relay 1 -
6	Relay 2 +
7	Relay 2 -
8	Relay 3 +
9	Relay 3 -
10	Relay 4 +
11	Relay 4 -
12	Relay 5 +
13	Relay 5 -

Table 9.5: Outputs of Environmental Controller Mapped to Relais

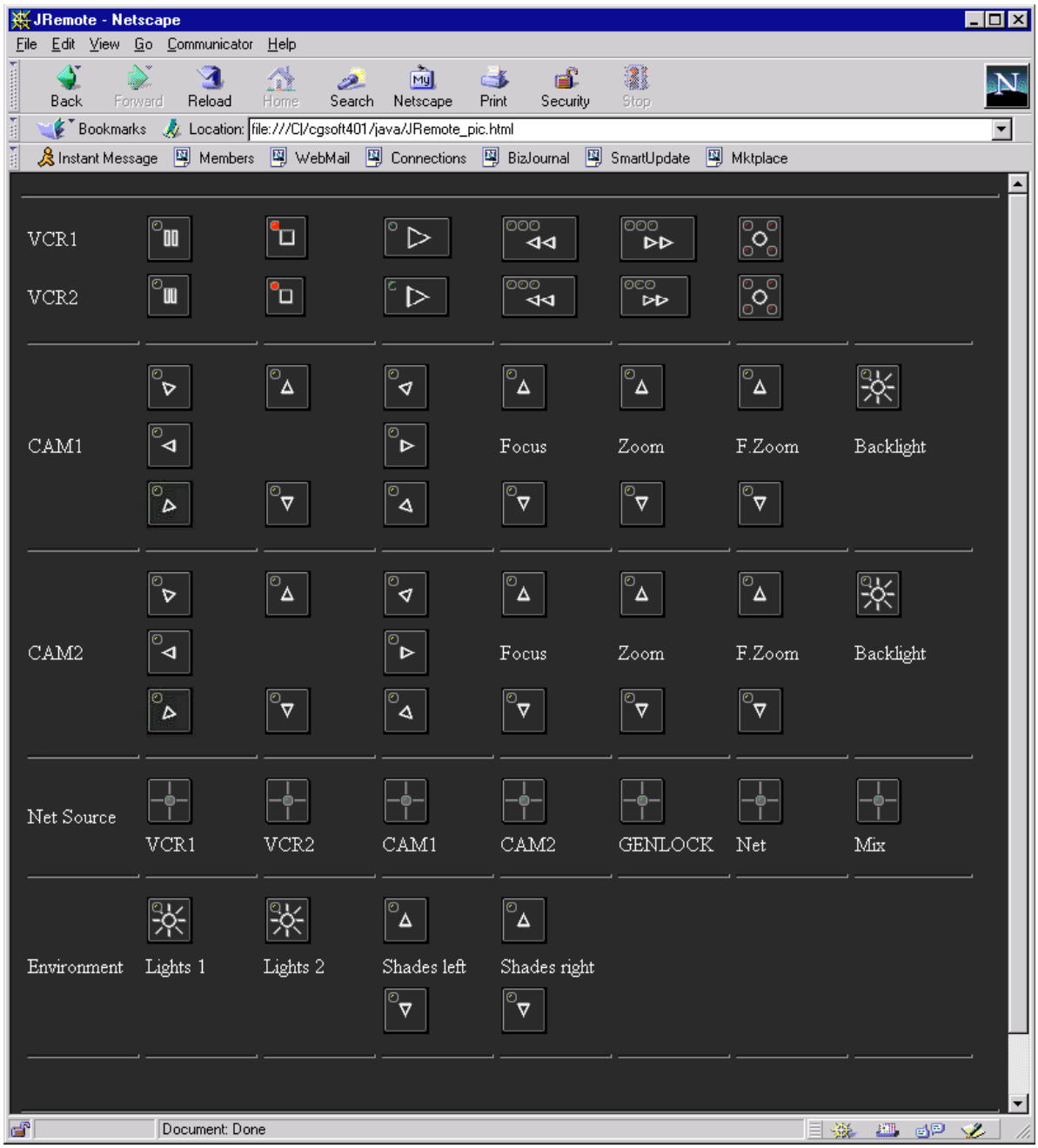

Figure 9.3: Screenshot of Java Remote Control Application

```
namedObject > [ t_SoBase *ret ]
   :  object > [ $ret ] %$

   |  << char*     id;
               t_SoBase* obj; >>
      DEF
      name   > [ id ]
      object > [ obj ]
              <<
               dataBase.addEntry(id,obj);
               if (id[0]=='_') {
                 defineObject(id, obj);
                 if (!v_showDefinition) {
                    $ret=NULL; %$
                    t_Object* o;
                    //ostack->pop(o);
                    cerr<<"Warning: this def will not be viewed:"<<id<<endl;
                  } else { $ret=obj; }
               } else { $ret=obj; }

               delete[] id;
               >>

   |           << char* id;
               t_SoBase* obj; >>
      USE
      name > [ id ]
            << obj=dataBase.getEntry(id);
               delete[] id;
               if (obj!=NULL)
               {
                  if  (v_useReferences) {
                    t_SoRef* ref=new t_SoRef;
                    ref->object = obj;
                    //cerr<<"Obj:"<<obj<<endl;
                    $ret=ref;  // nur referenz erzeugen
                  } else {
                    $ret=obj->clone();  // neue instanz erzeugen
                  }
               }
               else
               {
                  $ret=new t_SoUnknown("Unbound Name");
               } >>

;
```

Figure 9.4: Scene Grammar for the Modified VRML Parser

```
object[t_4x3Matrix* m]
   :        << t_SurfaceObject* obj;
               t_VolumeObject*  vol;
               int shdrnr=-1;
            >>
            << if (m!=NULL) startModTransf(*m);
            >>
            (  object_with_surface > [ obj ,  shdrnr ]
               << ostack->push(shdrnr,obj,sstack,bstack, dummySurfaceShader());
                  // Create area light source if surface is emissive
                  if (obj -> shader() -> emission() != t_Color::Black())
                     lstack -> push(new t_LightObject(obj));  >>
            BEGIN                              /* start sub-scene */
               << FloatVariables->begin();    VectorVariables->begin();
               MatrixVariables->begin();   ObjectVariables->begin();
               >>
               ( vardecl )*
               << startBound(); >>
               ( srf_shader | transf_obj )+
             END                                /* end sub-scene */
               << endBound();
                  FloatVariables->end();    VectorVariables->end();
                  MatrixVariables->end();   ObjectVariables->end();
               >>
          | INCLUDE                             /* include scene description */
               << char *fn; >>
               FILENAME                         /* msd filename */
               << include($2->getText()); >>  %$

          | REF                                 /* reference object */
               << char* name; >>
               ovar   > [ name ]
               { number > [ shdrnr ] }
               ";"
               << createReference(name, &acc_tr_matrix, shdrnr);
                  delete[] name;
               >>

          | EXT                                /* external reference */
               << char* name; char* extname; >>
                 var    > [ extname ]
                 ovar   > [ name ]
                 { number > [ shdrnr ] }  ";"
               << t_Object* obj= createReference(name, m, shdrnr);
                  //create external reference
                  ExternVariables->define(extname);
                 (*ExternVariables)[extname]=obj;
                  delete[] name;
                  delete[] extname;
               >>
         )
         << if (m!=NULL) { endModTransf();  delete m; }   >>
;
```

Figure 9.5: Scene Grammar for the Modified MSD Parser

Channel	Wires	Function
0	1,2	Shades down left
1	1,3	Shades down right
2	4,5	Shades up right
3	4,6	Shades up left
4	7,8	Lights 1
5	9,10	Lights 2

Table 9.6: Channel Description of Controller 1 (Front)

Channel	Function
0	Slides next
1	Slides direction, 0=forward, 1=back
2	Slide Sharpness increase
3	Slide Sharpness descrease
4	NC
5	Slide Projector Power

Table 9.7: Channel Description of Controller 2 (Back)

Bits	function
1000udlr	Bit Code
CAMLEFT	0x10001101
CAMRIGHT	0x10001110
CAMUP	0x10000111
CAMDOWN	0x10001011
CAMSTOP	0x10001111
CAMLEFTUP	0x10000101
CAMLEFTDOWN	0x10000110
CAMRIGHTUP	0x10001001
CAMRIGHTDOWN	0x10001010
IR Command	Byte Code
ToggleBackLite	40
ZoomInSlow	26
ZoomInFast	26
ZoomOutSlow	27
ZoomOutFast	29
FocusIn	34
FocusOut	35

Table 9.8: Control Codes for Camera Controller

ASCII String	Command
Projector Commands	
PROJECTOR*n* SWITCHTO VIDEO (ON/OFF))	Source Select
PROJECTOR*n* SWITCHTO RGB1 (ON/OFF))	Source Select
PROJECTOR*n* SWITCHTO RGB2 (ON/OFF))	Source Select
PROJECTOR*n* SWITCHTO SVIDEO (ON/OFF))	Source Select
PROJECTOR*n* ZOOM (IN/OUT) (ON/OFF))	optical Zoom
PROJECTOR*n* DIGIZOOM (IN/OUT) (ON/OFF))	Digital Zoom
PROJECTOR*n* FOCUS (IN/OUT) (ON/OFF))	Optical Focus
PROJECTOR*n* MOVESCREEN UP (ON/OFF)	Move digital Zoom image
PROJECTOR*n* MOVESCREEN DOWN (ON/OFF)	Move digital Zoom image
PROJECTOR*n* MOVESCREEN LEFT (ON/OFF)	Move digital Zoom image
PROJECTOR*n* MOVESCREEN RIGHT (ON/OFF)	Move digital Zoom image
PROJECTOR*n* MOVESCREEN UPRIGHT (ON/OFF)	Move digital Zoom image
PROJECTOR*n* MOVESCREEN UPLEFT (ON/OFF)	Move digital Zoom image
PROJECTOR*n* MOVESCREEN DOWNRIGHT (ON/OFF)	Move digital Zoom image
PROJECTOR*n* MOVESCREEN DOWNLEFT (ON/OFF)	Move digital Zoom image
PROJECTOR*n* POWER (ON/OFF)	Stand by
Camera Control	
CAMERA*n* FOCUS (IN/OUT) (ON/OFF)	Focus Control of Camera *n*
CAMERA*n* ZOOM (IN/OUT) {FAST} (ON/OFF)	Zoom Control of Camera *n*
CAMERA*n* BACKLIGHT	Toggle Backlight function
CAMERA*n* SCHWENK HOCH (ON/OFF)	Movement of Pan-Tilt Head
CAMERA*n* SCHWENK RUNTER (ON/OFF)	Movement of Pan-Tilt Head
CAMERA*n* SCHWENK RECHTS (ON/OFF)	Movement of Pan-Tilt Head
CAMERA*n* SCHWENK LINKS (ON/OFF)	Movement of Pan-Tilt Head
CAMERA*n* SCHWENK HOCH LINKS(ON/OFF)	Movement of Pan-Tilt Head
CAMERA*n* SCHWENK RUNTER LINKS(ON/OFF)	Movement of Pan-Tilt Head
CAMERA*n* SCHWENK HOCH RECHTS (ON/OFF)	Movement of Pan-Tilt Head
CAMERA*n* SCHWENK RUNTER RECHTS (ON/OFF)	Movement of Pan-Tilt Head

Table 9.9: ***Part I:*** **Control Commands Used for TCP/IP Connection (Adress 131.220.9.213, Port 5001) or Local Connections via the GUI. Parameters (ON|OFF) Used to Identify Keypress and Keyrelease Functions**

ASCII String	Command
Video Control	
REKORDER*n* PLAY	Control Recorder
REKORDER*n* PAUSE	Sill Image
REKORDER*n* STOP	Stop the Recorder
REKORDER*n* REWIND	picture search while PLAY
REKORDER*n* FORWARD	picture search while PLAY
REKORDER*n* RECORD	Record (manual source select)
Audio Control	
AUDIO MUTE *n*	Mute Channel *n*
AUDIO (PLUS/MINUS) *n*	inc., dec. Volume by one for Channel *n*
AUDIO (PLUS5/MINUS5) *n*	inc., dec Volume by five for Channel *n*
Environment	
SWITCHBOX1 1 (ON/OFF)	Shades left down
SWITCHBOX1 2 (ON/OFF)	Shades left up
SWITCHBOX1 3 (ON/OFF)	Shades right down
SWITCHBOX1 4 (ON/OFF)	Shades right up
SWITCHBOX1 5 (ON/OFF)	Lamps 1 power
SWITCHBOX1 6 (ON/OFF)	Lamps 2 power
SWITCHBOX2 12 (ON/OFF)	previous Slide
SWITCHBOX2 2 (ON/OFF)	next Slide
SWITCHBOX2 3 (ON/OFF)	Slide Focus in
SWITCHBOX2 4 (ON/OFF)	Slide Focus out
SWITCHBOX2 56 (ON/OFF)	Slide power
8x8 Matrix	
MATRIX *out in* (ON/OFF)	Switch input *in* to output *out*

Table 9.10: ***Part II:*** **Control Commands Used for TCP/IP Connections (Adress 131.220.9.213, Port 5001) or Local Connections via the GUI. Parameters (ON|OFF) Used to Identify Keypress and Keyrelease Functions**

Type	ID
Geo-Objects	
C_OBJECT_BOX	0x01010001
C_OBJECT_CONE	0x01010002
C_OBJECT_CYCOAT	0x01010003
C_OBJECT_DISC	0x01010004
C_OBJECT_ELLIPSOID	0x01010005
C_OBJECT_CYLINDER	0x01010006
C_OBJECT_PARALLEL	0x01010008
C_OBJECT_QUADRANGLE	0x01010009
C_OBJECT_REFOBJECT	0x0101000a
C_OBJECT_SPHERE	0x0101000b
C_OBJECT_SUPERQUAD	0x0101000c
C_OBJECT_TETRAHEDRON	0x0101000d
C_OBJECT_TORUS	0x0101000e
C_OBJECT_TRIANGLE	0x0101000f
C_OBJECT_SRPOLYLINE	0x01010010
C_OBJECT_SRBEZIER	0x01010012
C_OBJECT_SRTORUS	0x01010013
C_OBJECT_BPATCH	0x01010014
C_OBJECT_CSG	0x01010015
C_OBJECT_HFOBJ	0x01010016
C_OBJECT_TTF3D	0x01010017
C_OBJECT_3DSCAN	0x01010018
C_OBJECT_ISOSURF	0x01010019
C_OBJECT_SFBRID	0x0101001A
C_OBJECT_MBALL	0x0101001B

Table 9.11: Type Identifier of the Supported MRT Objects *Part I:*

Type	ID
Lights	
C_LIGHT_LIGHT	0x01040100
C_LIGHT_SPLIGH	0x01040101
C_LIGHT_ALIGHT	0x01040102
C_LIGHT_ASPLIG	0x01040103
C_LIGHT_DLIGHT	0x01040104
C_LIGHT_DSPLIG	0x01040105
C_LIGHT_ADSPLI	0x01040106
C_LIGHT_LIGOBJ	0x01040107
Surfaces	
C_SURF_SURFST	0x01020200
C_SURF_SURFFT	0x01020201
C_SURF_STCSPH	0x01020202
C_SURF_STWOOD	0x01020203
C_SURF_STNOIS	0x01020204
C_SURF_STTURB	0x01020205
C_SURF_STMARB	0x01020206
C_SURF_2T8B	0x01020207
C_SURF_2T24B	0x01020208
C_SURF_2T24BT	0x01020209
C_SURF_BUMP	0x0102020a
C_SURF_IBUMP	0x0102020b
C_SURF_SBUMP	0x0102020c
C_SURF_RBUMP	0x0102020d
C_SURF_PBUMP	0x0102020e
C_SURF_RIBUMP	0x0102020f
C_SURF_PIBUMP	0x01020210

Table 9.12: Type Identifier of the Supported MRT Objects*Part II:*

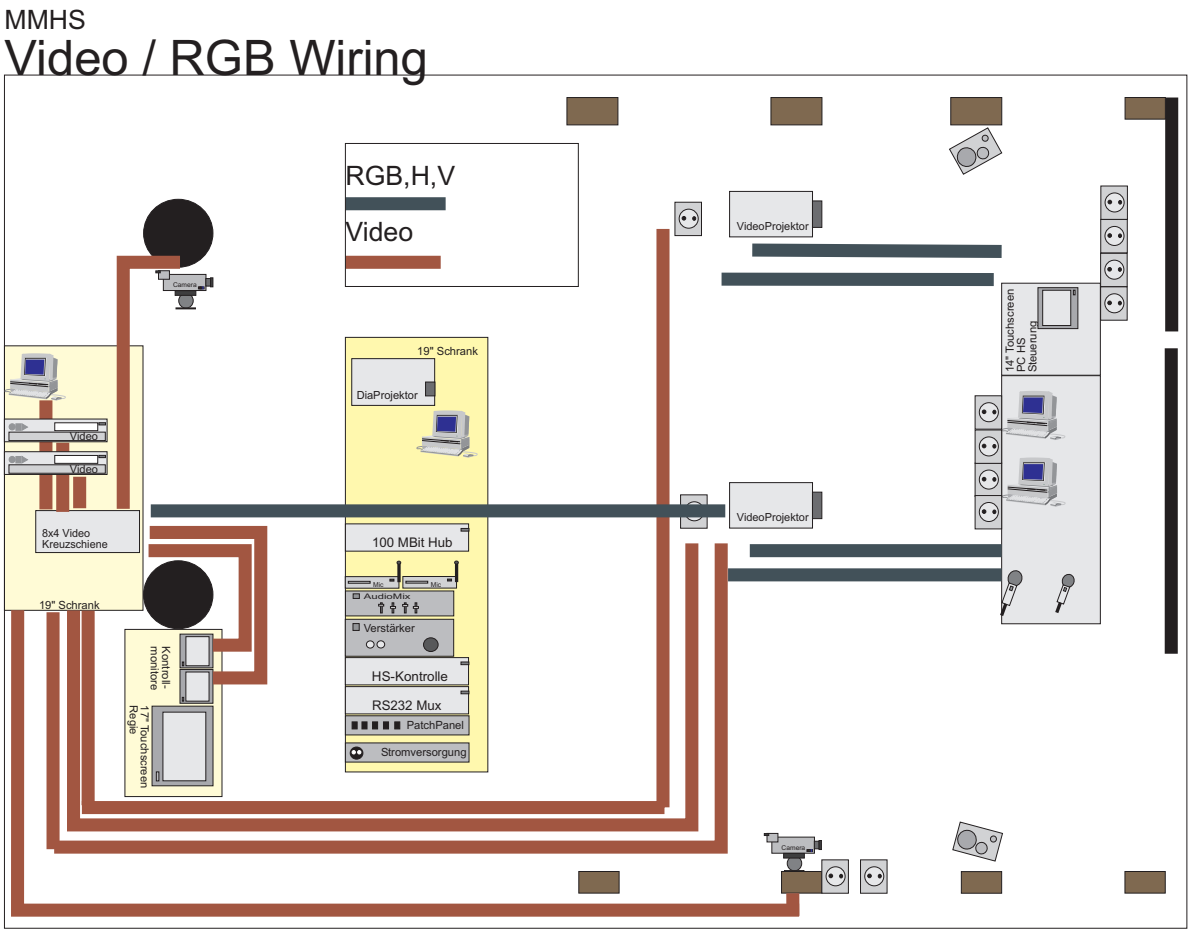

Figure 9.6: Video Wiring Diagram

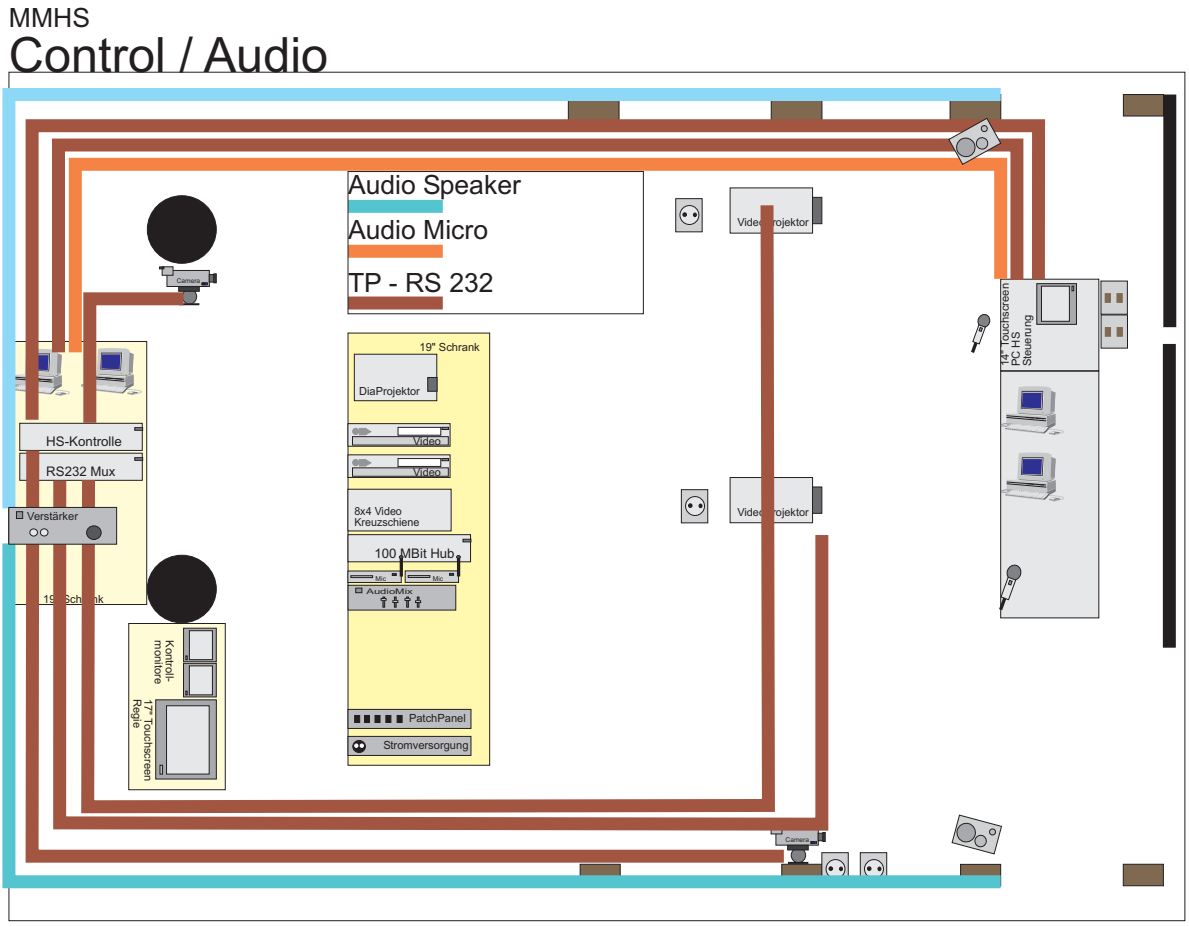

Figure 9.7: Wiring Diagram for Controllers

Bibliography

[Bar96] BARTKE R.: Interactive Multi-User Computer Games. *Internet, http://www/aid.wuien.ac.at/ mitloehn/mudreport.txt* (1996). pages 1

[BLFF96] BERNERS-LEE T., FIELDING R., FRYSYK H.: Hypertext Transfer Protocol - HTTP / 1.0 . *RFC 1945, May 1996* (1996). pages 11

[BPP95a] BELL G., PARISI A., PESCE M.: The virtual reality modeling language, version 1.0 specification. `http://www.sdsc.edu/SDSC/Partners/vrml/Archives/vrml10-3.html`, May 1995. pages 5

[BPP*95b] BELL G., PARISI A., PESCE M., MITRA, HARDENBERGH J. C., MEYER T., MARTIN B., CAREY R., MARBRY J., BLAU B.: The virtual reality modeling language, version 1.1 draft. `http://vag.vrml.org/vrml-1.1.html`, Dec. 1995. pages 5

[Bro97] BROLL W.: DWTP-An Internet Protocol for Shared Virtual Environments. *In Proceedings of the Virtual Reality Modeling Language Symposium 1998 (VRML98), Monterey, Ca., February 16-19, 1998 ,ACM SIGGRAPH* (1997), 49–56. pages 1, 14, 15

[BS98] BROLL W., SCHICK D.: DWTP-A Basis for Networked VR on the Internet. *In Proceedings of IST/SPIE's Symposium on Electronic Imaging: Science and Technology 1998 (EI'98), Stereoscopic Displays and Virtual Reality Systems V, (San Jose, January 24 - 30, 1998), M.T. Bolas, S. S. Fisher, and J. O. Merritt (eds.). SPIE, Bellingham, WA* (1998), 370–380. pages 14

[BZWM97] BRUTZMAN D., ZYDA M., WATSEN K., MACEDONIA M.: Virtual reality transfer protocol (vrtp) Design Rationale. *Workshops on Enabling Technology* (1997). pages 1, 7, 11

[CBS98] CREMERS A., BURGARD W., SCHULZ D.: Architecture of the Robotic Tele Lab. In *Proc. of the 1997 Annual Conference on Advances in Multimedia and Simulation* (Bochum, Germany, 1998). pages 73

[CMB96] CAREY R., MARRIN C., BELL G.: VRML EAI, External Authoring Interface, Part II of VRML 97 Standard. *International Standards Organisation (ISO(IEC)) draft Standard 14772, August 4 1996* (1996). pages 6

[Cor] CORPORATION M.: Microsoft Netmeeting homepage . *Internet, www.microsoft.com/netmeeting/*. pages 21

[Cor96a] CORPORATION M.: Microsoft DCOM homepage. *Internet, www.microsoft.com/dcom* (1996). pages 23

[Cor96b] CORPORATION N.: JavaScript Introduction . *Internet, http://home.netscape.com/ eng/mozilla/3.0/handbook/ javascript/introd.html* (1996). pages 2

[Cro96] CROWCROFT J.: Workshop on Matters MBone . *SIGCOMM Special Interest Grup on Data Communication - Palo Alto California - 16 Aug 1996* (1996). pages 9

[Dam96] DAMER B.: Inhabited digital spaces. *Internet, http://www.ccon.org/papers/chidemcc.html* (1996). pages 1

[DD95] DRUCKER S. M., D. Z.: CamDroid: A system for implementing intelligent camera control. In *Proc. of SIGGRAPH '95* (1995). pages 78

[Dee89] DEERING S.: Host Extension for IP Multicasting. *RFC 1112, August 1989, Internet, ftp://ds.internic.net/rfc/rfc1112.txt* (1989). pages 8

[Fed94] FEDOR C.: TCX - An interprocess communication system for building robotic architectures. *CMU Robotics Institute* (1994). pages 30

[Fel92] FELLNER D. W.: *Computer Grafik*, 2 ed., vol. 58 of *Reihe Informatik*. B.I. Wissenschaftsverlag, Mannheim, 1992. pages 46

[Fel96] FELLNER D. W.: Extensible image synthesis. In *Object-Oriented and Mixed Programming Paradigms*, Wisskirchen P., (Ed.), Focus on Computer Graphics. Springer, 1996, pp. 7–21. pages 22, 33

[FFW93] FELLNER D. W., FISCHER M., WEBER J.: *CGI-3D – A 3D Graphics Interface*. Tech. Rep. IAI-TR-95-x, University of Bonn, Dept. of Computer Science, Bonn, Germany, Aug. 1993. pages 46

[Fis96] FISCHER M.: *Software Architectures in Computer Graphics*. PhD thesis, University of Bonn, Dept. of Computer Science, Shaker Verlag, July 1996. ISBN 3-8265-2254-0. pages 41

[FJ96] FELLNER D., JUCKNATH O.: MRTSpace - Multi User Environments using VRML. *Proceedings of WebNet96* (1996). pages 35

[FS96] FELLNER D., SCHAEFER S.: MRT++ – Design Issues and Brief Reference. *IEEE Computer Graphics and Applications, Vol. 16, No. 3, May 1996, Internet, http://www.computer.org/cga/cg1999/g2079abs.htm* (1996). pages 41

[Gro94] GROUP O. O. M.: The Common Object Request Broker: Architecture and Specification. *Object Management Group Document* (1994). pages 18

[Gro96] GROUP W. W. W. C. W. W.: Hypertext Transfer Protocol - Next Generation . *Internet, www.w3.org/pub/WWW/Protocols/HTTP-NG* (1996). pages 11

[Han96] HANDLEY M.: Session Directory. *Internet, http://ugwww.ucs.ed.ac.uk/mice/archive/sdr.html* (1996). pages 22, 38

[HCS96] HE L.-W., COHEN M. F., SALESIN D. H.: The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In *Proc. of SIGGRAPH '96* (1996). pages 78

[HF96] HOPP A., FELLNER D. W.: MRT-VR - Verteilte dynamische Multi-User-3D Umgebung. *AAA Conference, IGD Darmstadt, Okt* (1996). pages 3

[HF97] HOPP A., FELLNER D. W.: MRT-VR a Multi User Environment. *GI Jahrestagung Aachen* (1997). pages 3

[HF98] HOPP A., FELLNER D. W.: MRT-VR - A Multi User Environment for Education and Simulation. *SCS Conference, ED MEDIA, Leicester UK, Jan 1998* (1998). pages 3, 22

[HF99] HOPP A., FELLNER D. W.: VR-Lab - A Multi User Environment for Educational Purposes. *in Conference Procedings VRML 99, Paderborn, Germany* (1999). pages 3

[Hop97] HOPP A.: MBP - MRT Binary Protocol. *Internet, www.informatik.uni-bonn.de/hopp/MRT-VR.html* (1997). pages 34

[Inc97] INC. M. T.: PIC 1657 Technical Reference. *1997 Technical Library First Edition* (1997). pages 61

[ISO89] ISO: *Information Processing Systems – Computer Graphics – Interfacing techniques for dialogues with graphical devices (CGI), Part 1-6,* DIS 9636, Dec. 1989. pages 46

[JMa] JACKOBSON V., MCCANNE S.: Visual Audio Toolkit. *Internet, ftp.ee.lbl.gov, Lawrence Berkeley Laboratory,University of California, Berkeley, CA.*. pages 21

[JMb] JACKOBSON V., MCCANNE S.: Whiteboard. *Internet, ftp.ee.lbl.gov,Lawrence Berkeley Laboratory, University of California, Berkeley, CA.*. pages 21

[Lau] LAUKIEN M.: Dokumentation ORBacus Version 3.1.1. *Object Oriented Concepts, Inc., www.ooc.com*. pages 19

[Leo96] LEONHARD R.: Untersuchung über 3D Viewer. *Seminar a.d. Rheinischen Freidrich Wilhelms Universität Bonn* (1996). pages 1

[Leo99] LEONHARD R.: Entwicklung einer Datenhaltungsschicht für virtuelle Welten. *Diplomarbeit der Universität Bonn* (1999). pages 23, 31

[Mac95] MACDONIA M.: A Network Software Architecture for Large Scxale Virtual Envormonments . *Ph.D. Dissertation, Naval Postgraduate Scool, Monterey, California, June 1995* (1995). pages 9

[Mau96] MAURER H. (Ed.): *Hyper-G/now Hyperwave – The Next Generation Web Solution*. Addison-Wesley, Harlow, England, 1996. pages 50

[Mau98] MAUVE M.: Transparent Access To And Encoding of VRML State Information. *in Proceendings of VRML99, 4th Symposium on VRML* (1998), 29–39. pages 6

[MB94] MACEDONIA M., BRUTZMANN D. P.: MBone provides Video and Audio across the Internet. *IEE Transactions on Computers* (Apr. 1994). pages 30

[MB95] MACEDONIA M., BRUTZMANN D. P.: MPSNET: A Multi Player 3D Virtual Environment over the Internet. *Symposion in interactive 3D graphics* (1995). pages 31

[MF99] MÜLLER G., FELLNER D.: Hybrid Scene Structuring with Application to Ray Tracing. . *Proceedings of ICVC '99, Goa, India, Feb.* (1999). pages 44

[MJ95] MCCANNE S., JACOBSON V.: VIC - A flexible framework for packet video. *Proceedings of ACM Multimedia* (1995). pages 2, 21

[Nie93] NIELSEN J.: Usability Engineering. *ap, 1300 boylston Street, Chestnut Hill, MA 02167* (1993). pages 56, 57, 58, 60

[Obl95] OBLINGER D.: Educational Alternatives based on communication, Collaboration and Computers. *Internet, www.cba.uh.edu/inetcrse/internet/ideas.html* (1995). pages 1

[Par95] PARR T.: Language Translation Unsing PCCTS and C++. *Reference Guide, Automata Publishing Cpmpany, 1072 South DeAnza Blvd, Auite A107, San Jose California, 95129 USA, ISBN 0-9627488-5-4* (1995). pages 45

[Pul96] PULKKA A.: Spatial Culling of interpersonal Communication within large scale Multi-User virtual Environments. *Internet, www.hitl.washington.edu/publications/pulkka/abstract.html* (1996). pages 1, 30

[RTS98] REINEMA R., TIETZE D., STEINMETZ R.: IMCE - An Integrated Multimedia Conferencing and Collaboration Environment. *Proceedings of the First National CSCW Workshop (CCSCW98)„ ISBN 7-5053-5066-8, Publishing House of Electronics Industry, Beijing, China*, (1998), 95–100. pages 1

[Saa98] SAAR K.: Virtus: A Collaborative Multi-User Platform. *in Proceendings of VRML99, 4th Symposium on VRML* (1998), 141–152. pages 7, 22

[Sch96] SCHULZRINNE H.: RTP Profile for Audio and Video Conferences with minimal Control . *Audio-Video Transport WG, RFC 1890* (1996). pages 8

[Sch98] SCHULZ B. C. F. H.: Virtual Reality Visualization of Distributed Tele-Experiments. *IECON 1998* (1998). pages 73

[Sil93] SILICON GRAPHICS INC.: *The OpenGL Reference Manual – The Official Reference Document for OpenGL*, 1 ed. Addison-Wesley, Reading, Mass., 1993. pages 21

[Sto79] STONEBRAKER M.: Concurrency Control and Cosistency of Multiple Copies. *in Distributed Ingres. IEEE Trans. on Software Engineering 5 (3), 188-194* (1979). pages 24

[Stu98] STURZEBECHER D.: MACS - A flexible and scalable collaboration Environment. *ED Media, http://www.ibr.cs.tu-bs.de/projects/macs* (1998). pages 1, 1, 1

[Sun93] SUN MICROSYSTEMS: *The Solaris XGL Graphics Library*. Sun Microsystems Inc., Mountain View, CA, Feb. 1993. pages 1, 21

[Swe98] SWEENEY T.: Unreal Networking Architecture. *Internet, www.epicgames.com* (1998). pages 16

[TBB*98] THRUN S., BÜCKEN A., BURGARD W., FOX D., FRÖHLINGHAUS T., HENNIG D., HOFMANN T., KRELL M., SCHIMDT T.: Map learning and high-speed navigation in RHINO. In *AI-based Mobile Robots: Case studies of successful robot systems*, Kortenkamp D., Bonasso R.„ Murphy R., (Eds.). MIT Press, Cambridge, MA, 1998. pages 73, 74

[Tie97] TIE L.: WebCanal: a Multicast Web Application . *INRIA Rocquencourt, BP 105* (1997). pages 12

[WMK96] WHETTEN B., MONTGOMERY T., KAPLAN S.: High Performance Totally Ordered Multicast Protocol. *Theory and Practice in Distributed Systems, Springer Verlag LCNS 938* (1996). pages 15